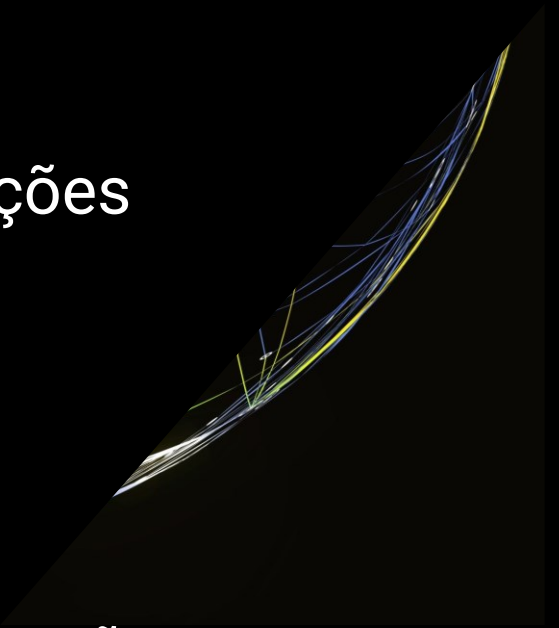
COMPUTAÇÃO GRÁFICA

Aula 4 – Transformações Geométricas



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

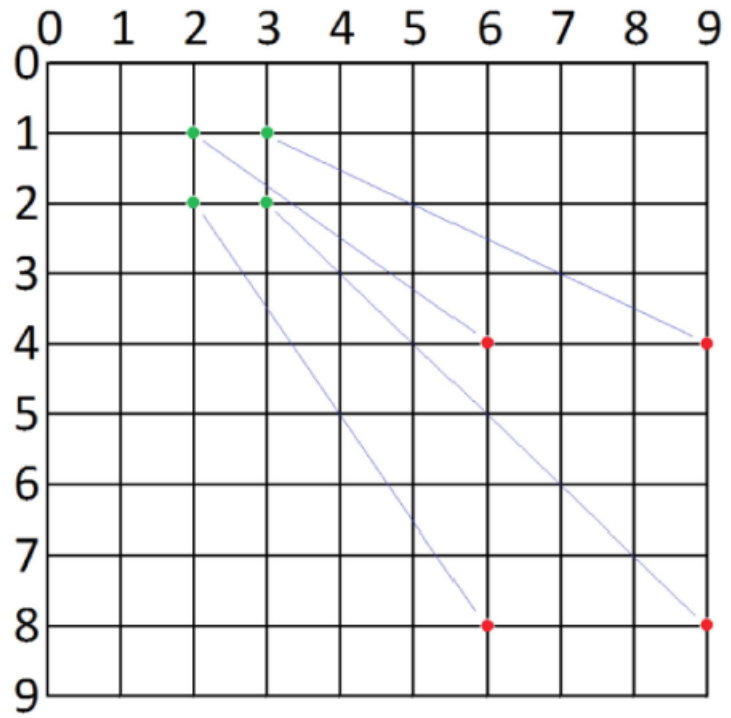
Caro aluno, mais uma vez vamos enfatizar a importância da geometria para o processamento gráfico. Toda animação gráfica depende da aplicação de transformações geométricas sobre imagens digitais ou vetoriais ou sobre modelos tridimensionais. Desde um simples efeito de minimização de janela do seu sistema operacional até a movimentação de uma câmera em um jogo digital são produzidos por transformações geométricas.

As transformações geométricas são implementadas na forma de multiplicação de matrizes. Isto explica porque são tão computacionalmente complexas tais operações, e porque é comum a adoção de uma placa de vídeo dedicada em computadores pessoais. Esta seção aborda os conceitos teóricos das transformações geométricas e é a base para quem deseja implementar aplicativos com qualquer tipo de animação.

Translação e escala

A Figura 2.1 mostra a transformação de escala de quatro pontos: (2,1), (3,1), (2,2), (3,2). A escala aplicada foi de três vezes, no eixo x, e quatro vezes, no eixo y: $(s_x,s_y)=(3,4)$.

Figura 2.1 | Transformação de escala. Em verde os pontos originais e em ver após transformação



A transformação de escala de um ponto (x_1,y_1), por uma escala (s_x,s_y), se dá pelas equações

$$x_2 = s_x x_1$$

$$y_2 = s_y y_1$$

que podem ser representadas na forma de matriz como

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$$

E pode ser estendido para três dimensões como

$$\begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix}$$

Pela representação na forma de matriz, podemos sempre considerar que o ponto p_2 é resultado da transformação geométrica sobre p_1 pela equação $p_2 = M \cdot p_1$, onde M é a **matriz de transformação** (BOLDRINI et al., 1986). Assim é possível observar também que as transformações são reversíveis aplicando-se a matriz inversa: $p_1 = M^{-1} \cdot p_2$.

Assimile

Toda transformação geométrica de um conjunto de pontos é a execução de uma multiplicação de matrizes do tipo p2 = Mp1 para cada ponto do conjunto. Isso explica porque a computação gráfica exige um hardware dedicado somente ao processamento gráfico: a placa de vídeo.

Conceitos:

1. **Ponto homogêneo:**

   Um ponto homogêneo é uma representação de coordenadas em um espaço projetivo, usada frequentemente em transformações geométricas 2D e 3D. Em 2D, um ponto (x, y) pode ser representado como $(x, y, 1)$ no espaço homogêneo. Em 3D, um ponto (x, y, z) seria $(x, y, z, 1)$.

2. **Matriz de transformação:**

   A matriz de transformação é usada para aplicar operações como translação, rotação, escala ou até projeção a pontos no espaço.

   Por exemplo, uma matriz de transformação 2D pode ter a forma:

$$T = \begin{bmatrix} a & b & tx \\ c & d & ty \\ 0 & 0 & 1 \end{bmatrix}$$

   onde tx e ty são as translações em x e y, e os elementos a, b, c, d podem representar rotação, escala, ou cisalhamento.

3. **Multiplicação de matriz:**

   A operação `np.dot(transformation_matrix, homogeneous_point)` aplica a multiplicação de matriz entre a matriz de transformação (que pode ser 2D ou 3D) e o ponto homogêneo.

   Se T for uma matriz $3x3$ de transformação 2D e $P_h = [x, y, 1]$ for o ponto homogêneo, o resultado seria um novo ponto transformado no espaço 2D.

Explicação da linha de código:

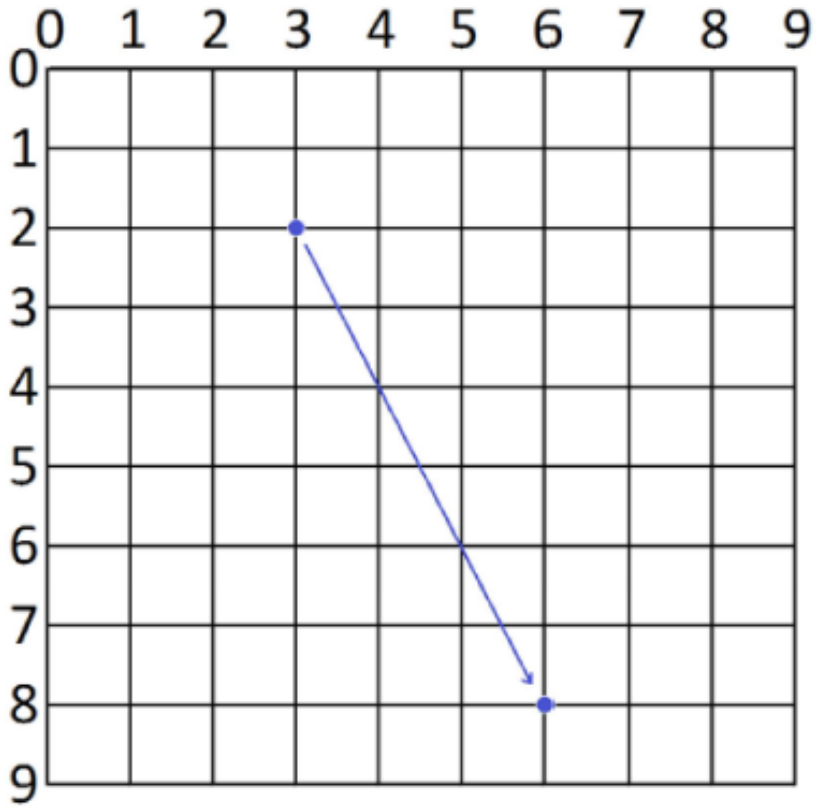
`transformation_matrix`: É a matriz que contém as operações (rotação, escala, translação etc.) que você quer aplicar ao ponto.

`homogeneous_point`: É o ponto em coordenadas homogêneas, geralmente algo como $[x, y, 1]$ em 2D ou $[x, y, z, 1]$ em 3D.

`np.dot`: A função `np.dot` do NumPy realiza a multiplicação de matriz entre a matriz de transformação e o ponto homogêneo. O resultado será o ponto transformado.

A segunda transformação mais simples é a translação. A Figura 2.2 representa por uma seta a translação do ponto *(x1,y1)=(3,2)* de *(tx, ty) = (3,6)*, levando-o para o ponto *(x2,y2)=(6,8)*.

Figura 2.2 | Translação do ponto *(3,2)* de uma distância *(3,6)*, levando-o para o ponto *(6,8)*



A translação de um ponto *(x1,y1)* por *(tx,ty)*, se dá pelas equações

$$x_2 = x_1 + t_x$$

$$y_2 = y_1 + t_y$$

Que podem ser representadas na forma de matriz como

2D: $$\begin{pmatrix} x_2 \\ y_2 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix},$$

3D: $$\begin{pmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{pmatrix}$$

Para a translação foi necessário acrescentar uma dimensão, fazendo com que a matriz de transformação 2D seja 3x3 e a matriz de transformação 3D seja 4x4. Da mesma forma podemos acrescentar uma dimensão na transformação de escala.

2D: $$\begin{pmatrix} x_2 \\ y_2 \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix},$$

3D: $$\begin{pmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{pmatrix}$$

De forma que toda matriz de transformação, seja de escala, translação ou rotação, sejam de dimensão 3x3 no caso 2D e 4x4 no caso 3D. Assim podemos finalmente definir as matrizes de escala e translação por

2D:

$$S = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad T = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix}$$

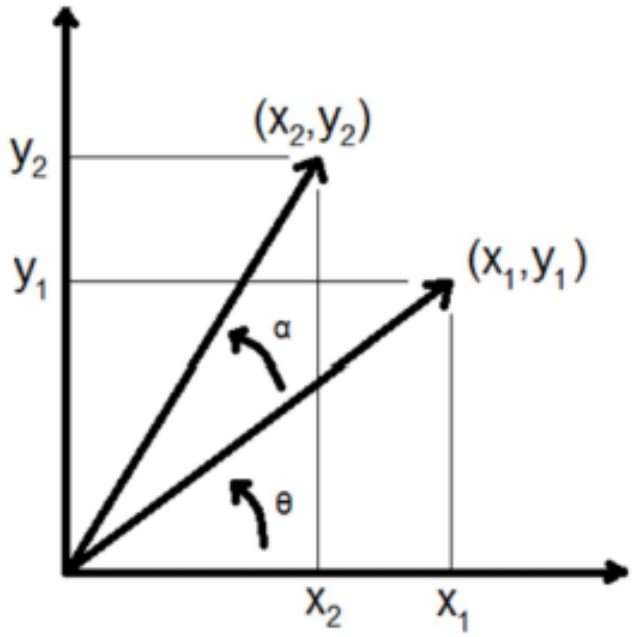
3D:

$$S = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad T = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotação

A rotação de um ponto bidimensional (x_1,x_2) por um ângulo α em torno da origem é representada na Figura 2.3.

Figura 2.3 | Rotação de (x_1,y_1) de um ângulo α em torno da origem, levando-o para o ponto (x_2,y_2)



Com base nos conceitos da geometria plana e relações trigonométricas (BOLDRINI et al. 1986; KREYSZIG, 2015), vemos que

$$x_2 = r\cos(\theta+\alpha) = r\cos\theta\cos\alpha - r\sin\theta\sin\alpha$$

$$y_2 = r\sin(\theta+\alpha) = r\sin\theta\cos\alpha + r\cos\theta\sin\alpha$$

mas $x_1 = r\cos\theta, y_1 = r\sin\theta$, então

$$x_2 = x_1\cos\alpha - y_1\sin\alpha$$

$$y_2 = x_1\sin\alpha + y_1\cos\alpha$$

Logo, a matriz de rotação 2D em torno da origem é

$$R_\alpha = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

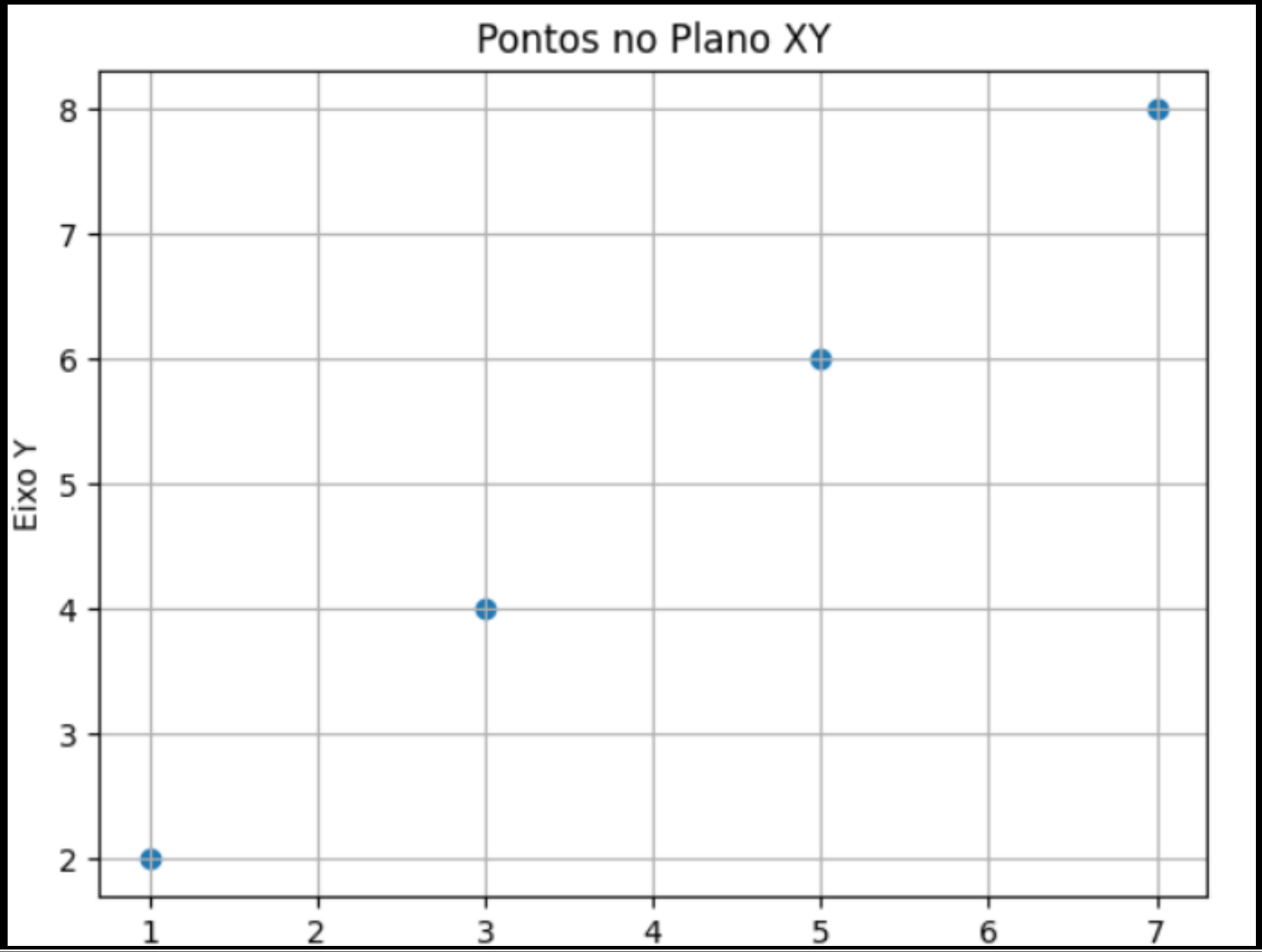
A matriz R pode ser facilmente estendida para três dimensões para as rotações em torno de um dos três eixos. Por ser uma rotação em torno do eixo Z, a coordenada Z não é alterada. A matriz de rotação é igual à matriz 2D, apenas acrescida da dimensão Z.

$$R_{z\alpha} = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Da qual podemos derivar $R_{x\alpha}$ e $R_{y\alpha}$ de rotação em torno dos eixos X e Y, respectivamente.

$$R_{x\alpha} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad R_{y\alpha} = \begin{pmatrix} \cos\alpha & 0 & -\sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ \sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

```python
# prompt: código para desenhar n pontos fornecidos a paritr de uma lista em um plano xy

import matplotlib.pyplot as plt

def plot_points(points):
  """
  Desenha pontos em um plano XY a partir de uma lista de coordenadas.

  Args:
    points: Uma lista de tuplas (x, y) representando as coordenadas dos pontos.
  """
  x_coords = [point[0] for point in points]
  y_coords = [point[1] for point in points]

  plt.scatter(x_coords, y_coords)
  plt.xlabel("Eixo X")
  plt.ylabel("Eixo Y")
  plt.title("Pontos no Plano XY")
  plt.grid(True)
  plt.show()

# Exemplo de uso:
points = [(1, 2), (3, 4), (5, 6), (7, 8)]
plot_points(points)
```


Pontos no Plano XY
Eixo Y
8
7
6
5
4
3
2
1
2
3
4
5
6
7

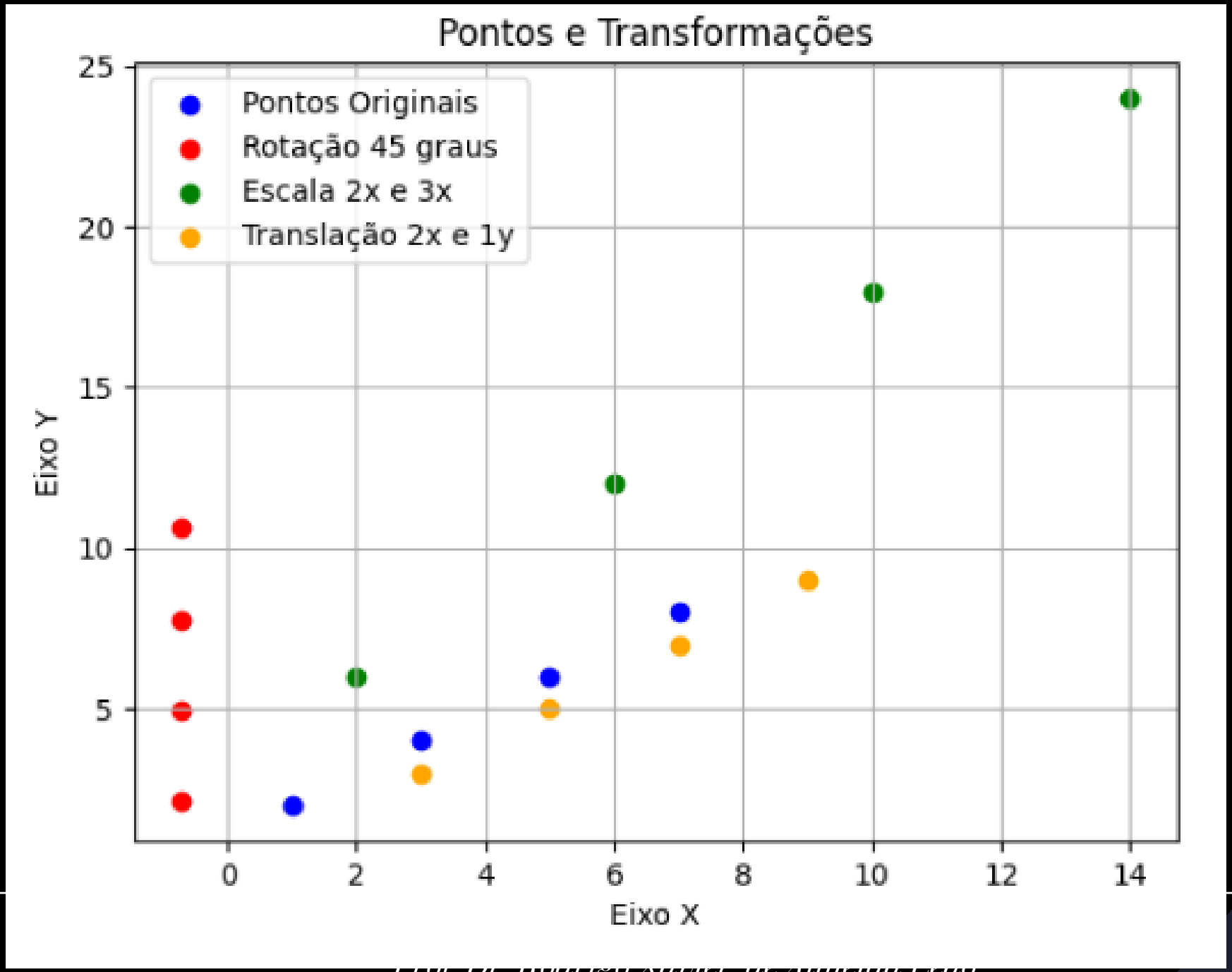
```python
import matplotlib.pyplot as plt
import numpy as np


def plot_points_with_lines(points):

    x_coords = [point[0] for point in points]
    y_coords = [point[1] for point in points]

    plt.plot(x_coords + [x_coords[0]], y_coords + [y_coords[0]], marker='o')
    plt.xlabel("Eixo X")
    plt.ylabel("Eixo Y")
    plt.title("Pontos Conectados por Linhas")
    plt.grid(True)
    plt.show()

# Exemplo de uso:
points = [(1, 2), (7, 2), (4, 5)]
plot_points_with_lines(points)

```


Pontos Conectados por Linhas
Eixo Y
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1
2
3
4
5
6
7
Eixo X

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

def plot_points(points, color='blue', label='Pontos Originais'):

  x_coords = [point[0] for point in points]
  y_coords = [point[1] for point in points]

  plt.scatter(x_coords, y_coords, color=color, label=label)

def apply_transformation(points, transformation_matrix):

  transformed_points = []
  for point in points:
    # Converter o ponto em um vetor coluna com coordenadas homogêneas (x, y, 1)
    if transformation_matrix.shape == (2, 2):
      homogeneous_point = np.array([point[0], point[1]])
      transformed_point = np.dot(transformation_matrix, homogeneous_point)
    elif transformation_matrix.shape == (3, 3):
      homogeneous_point = np.array([point[0], point[1], 1])
      transformed_point = np.dot(transformation_matrix, homogeneous_point)
      transformed_point = transformed_point[:2] / transformed_point[2]
    else:
      raise ValueError("A matriz de transformação deve ser 2x2 ou 3x3.")
    transformed_points.append((transformed_point[0], transformed_point[1]))
  return transformed_points
```

```python
# Exemplo de uso:
points = [(1, 2), (3, 4), (5, 6), (7, 8)]
plot_points(points)

# Matriz de transformação para rotação de 45 graus
angle = np.radians(45)
rotation_matrix = np.array([
    [np.cos(angle), -np.sin(angle)],
    [np.sin(angle), np.cos(angle)]  ])

transformed_points = apply_transformation(points, rotation_matrix)
plot_points(transformed_points, color='red', label='Rotação 45 graus')

# Matriz de transformação para escala de 2x no eixo x e 3x no eixo y
scaling_matrix = np.array([
    [2, 0],
    [0, 3]  ])

transformed_points = apply_transformation(points, scaling_matrix)
plot_points(transformed_points, color='green', label='Escala 2x e 3x')
```

```python
# Matriz de transformação para translação de 2 unidades no eixo x e 1 unidade no eixo y
translation_matrix = np.array([
    [1, 0, 2],
    [0, 1, 1],
    [0, 0, 1]  ])

transformed_points = apply_transformation(points, translation_matrix)
plot_points(transformed_points, color='orange', label='Translação 2x e 1y')

plt.xlabel("Eixo X")
plt.ylabel("Eixo Y")
plt.title("Pontos e Transformações")
plt.grid(True)
plt.legend()
plt.show()
```


Pontos e Transformações
Pontos Originais
Rotação 45 graus
Escala 2x e 3x
Translação 2x e 1y
Eixo Y
Eixo X
25
20
15
10
5
0
2
4
6
8
10
12
14