

プログラムリスト 7.1 NumPyでベクトルの内積と外積を計算する

In [38]:

```
import numpy as np                # (A1) NumPyライブラリをimport

a = np.array([2, 1, 3])           # (B1) ベクトルaを定義
b = np.array([1, -1, 0])          # (B2) ベクトルbを定義

norm_a = np.linalg.norm(a)        # (C1) ベクトルaの大きさ (ノルム) を計算
norm_b = np.linalg.norm(b)        # (C2) ベクトルbの大きさ (ノルム) を計算

dot_ab = np.dot(a, b)             # (C3) 内積を計算
cross_ab = np.cross(a, b)         # (C4) 外積を計算

print(f"norm of vector a: {norm_a:.5f}") # (D1) ベクトルaの大きさを表示
print(f"norm of vector b: {norm_b:.5f}") # (D2) ベクトルbの大きさを表示
print(f"inner product (a,b): {dot_ab}")  # (D3) 内積を表示
print(f"cross product (a,b): {cross_ab}") # (D4) 外積を表示
```

```
norm of vector a: 3.74166
norm of vector b: 1.41421
inner product (a,b): 1
cross product (a,b): [ 3  3 -3]
```

プログラムリスト 7.2 2次元図形の対称移動

In [39]:

```
import numpy as np                # (A1) numpyライブラリをimport
import matplotlib.pyplot as plt  # (A2) matplotlibライブラリをimport

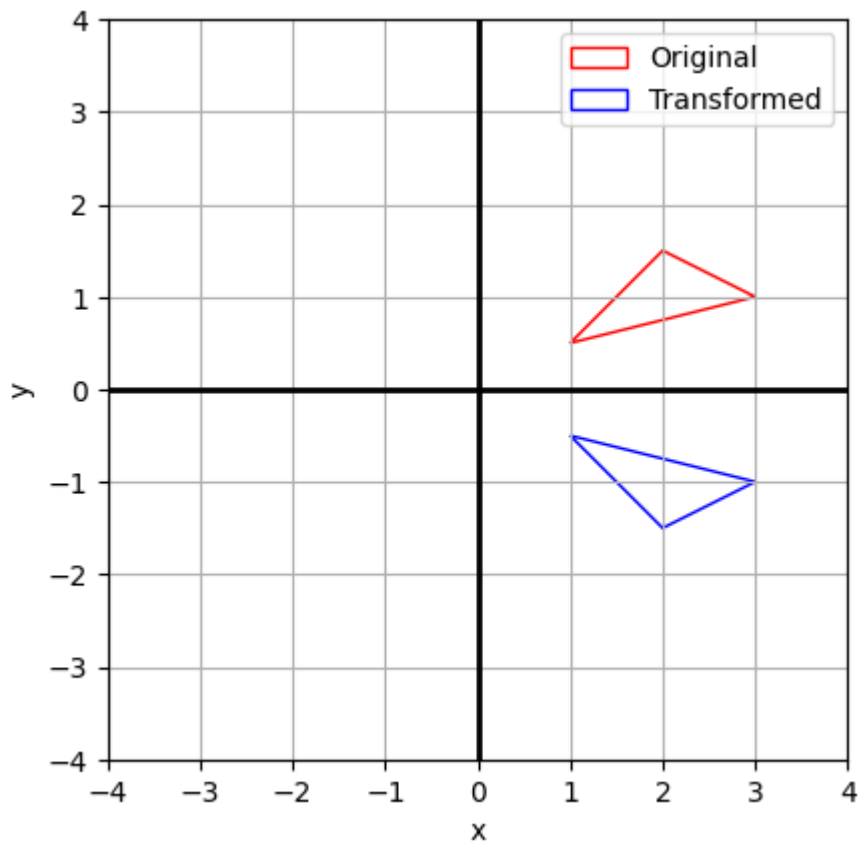
vertices = np.array([[1, 0.5],    # (B1) 三角形の頂点座標を定義 (各行が1頂点)
                     [2, 1.5],
                     [3, 1]])

trans_matrix = np.array([[1, 0],   # (B2) X軸対称移動の行列Sx (y座標の符号を反転)
                        [0, -1]])

verticesT = vertices.T            # (C1) 頂点行列を転置
trans_verticesT = trans_matrix @ verticesT # (C2) 行列積で一括変換
trans_vertices = trans_verticesT.T # (C3) 転置して座標形式に戻す

plt.fill(vertices[:, 0], vertices[:, 1], facecolor='none',
          edgecolor='red', label="Original") # (E1) 元の三角形 (赤)
plt.fill(trans_vertices[:, 0], trans_vertices[:, 1], facecolor='none',
          edgecolor='blue', label="Transformed") # (E2) 変換後三角形(青)

plt.grid(True)                   # (F1) グリッド線を表示
plt.xlabel("x")                  # (F2) x軸ラベルの設定
plt.ylabel("y")                  # (F3) y軸ラベルの設定
plt.axvline(x=0, color='black', linewidth=2) # (F4) y軸の線を太く表示
plt.axhline(y=0, color='black', linewidth=2) # (F5) x軸の線を太く表示
plt.legend()                     # (F6) 凡例を表示
plt.xlim(-4, 4)                  # (F7) x軸の範囲を設定
plt.ylim(-4, 4)                  # (F8) y軸の範囲を設定
plt.gca().set_aspect('equal')    # (F12) x軸とy軸のスケールを揃える
plt.show()                       # (G) グラフを表示
```



プログラムリスト 7.3 2次元図形の回転移動

```
In [40]: import numpy as np                # (A1) numpyライブラリをimport
import matplotlib.pyplot as plt        # (A2) matplotlibライブラリをimport

vertices = np.array([[1, 0.5],         # (B1) 三角形の頂点座標を定義 (各行が1頂点)
                    [2, 1.5],
                    [3, 1]])

theta = np.pi/2                       # (B2) 回転角 (ラジアン) = pi/2
trans_matrix = np.array([[np.cos(theta), -np.sin(theta)], # (B3) 回転行列R
                        [np.sin(theta), np.cos(theta)]])

verticesT = vertices.T                 # (C1) 頂点行列を転置
trans_verticesT = trans_matrix @ verticesT # (C2) 行列積で一括変換
trans_vertices = trans_verticesT.T      # (C3) 転置して座標形式に戻す

plt.fill(vertices[:, 0], vertices[:, 1], facecolor='none',
          edgecolor='red', label="Original") # (E1) 元の三角形 (赤)
plt.fill(trans_vertices[:, 0], trans_vertices[:, 1], facecolor='none',
          edgecolor='blue', label="Transformed") # (E2) 回転後 (三角形, 青)

plt.grid(True)                        # (F1) グリッド線を表示
plt.xlabel("x")                       # (F2) x軸ラベルの設定
plt.ylabel("y")                       # (F3) y軸ラベルの設定
plt.axvline(x=0, color='black', linewidth=2) # (F4) y軸の線を太く表示
plt.axhline(y=0, color='black', linewidth=2) # (F5) x軸の線を太く表示
plt.legend()                          # (F6) 凡例を表示
plt.xlim(-4, 4)                      # (F7) x軸の範囲を設定
plt.ylim(-4, 4)                      # (F8) y軸の範囲を設定
plt.gca().set_aspect('equal')         # (F12) x軸とy軸のスケールを揃える
plt.show()                           # (G) グラフを表示
```

