```
# python libraries
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import animation
from IPython.display import HTML
from math import pi
import plotly.graph_objects as go
import mpl_toolkits.mplot3d.axes3d as p3
import scipy
```

Q.1 Make a function that converts ZYX Euler angle representation to SO(3) [5 pts]

Complete the fuction returning the orientation matrix (SO(3)) given the Euler ZYX angles.

```
def ZYX_to_SO3(th_z, th_y, th_x):
  # Fill your code here
  Rz = np.array([
      [np.cos(th_z), -np.sin(th_z), 0],
      [np.sin(th_z), np.cos(th_z), 0],
      [0, 0, 1]])
  Ry = np.array([
      [np.cos(th_y), 0 ,  np.sin(th_y)],
      [0, 1, 0],
      [-np.sin(th_y),0, np.cos(th_y)]])
  Rx = np.array([
      [1, 0, 0],
      [0, np.cos(th_x), -np.sin(th_x)],
      [0, np.sin(th_x), np.cos(th_x)]])

  SO3 = Rz@Ry@Rx #make sure you use matrix multiplication not element wise multiplication "*"
  return SO3
```

Test function for ZYX_to_SO3. Use this code for debugging

```
def test_ZYX_to_SO3():
  zyx_test = np.array([[0.3, 0.2, 0.7],
                       [0.7, np.pi, np.pi/2],
                       [np.pi/3, 0, 0]])
  soln = np.array([[[ 0.93629336, -0.10375634,  0.33554338],
                  [ 0.28962948,  0.76850419, -0.57054017],
                  [-0.19866933,  0.63137622,  0.74959627]],
                 [[-7.64842187e-01,  5.42191972e-17,  6.44217687e-01],
                  [-6.44217687e-01,  1.25726990e-16, -7.64842187e-01],
                  [-1.22464680e-16, -1.00000000e+00, -6.12323400e-17]],
                 [[ 0.5       , -0.8660254,  0.        ],
                  [ 0.8660254,  0.5       ,  0.        ],
                  [ 0.        ,  0.       ,  1.        ]]])
  res = []
  for test in zyx_test:
    res.append(ZYX_to_SO3(*test))
  res = np.array(res)
  if np.allclose(res, soln):
    print('your implementation is correct')
  else:
    print('In correct implementation try again')
```

```
test_ZYX_to_SO3()
```

```
    your implementation is correct
```

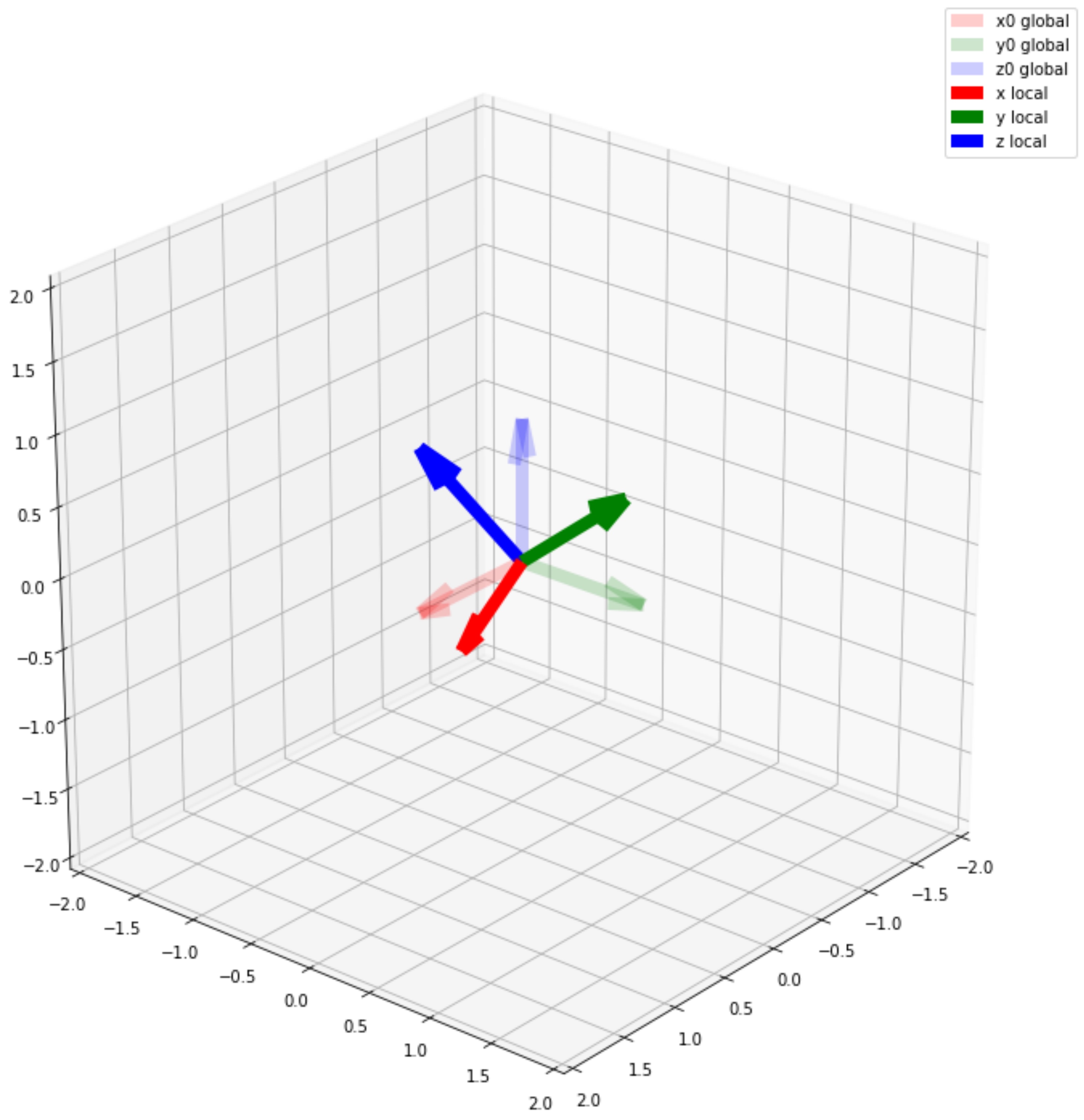
Q.2 Visualize the following frame orientations. [5 pts]

$$(a) ZYX = (0.3, 0.2, 0.7)\ [rad]$$
$$(b) ZYX = (0.7, 0, \pi/2)\ [rad]$$
$$(c) ZYX = (\pi/3, 0, 0)\ [rad]$$

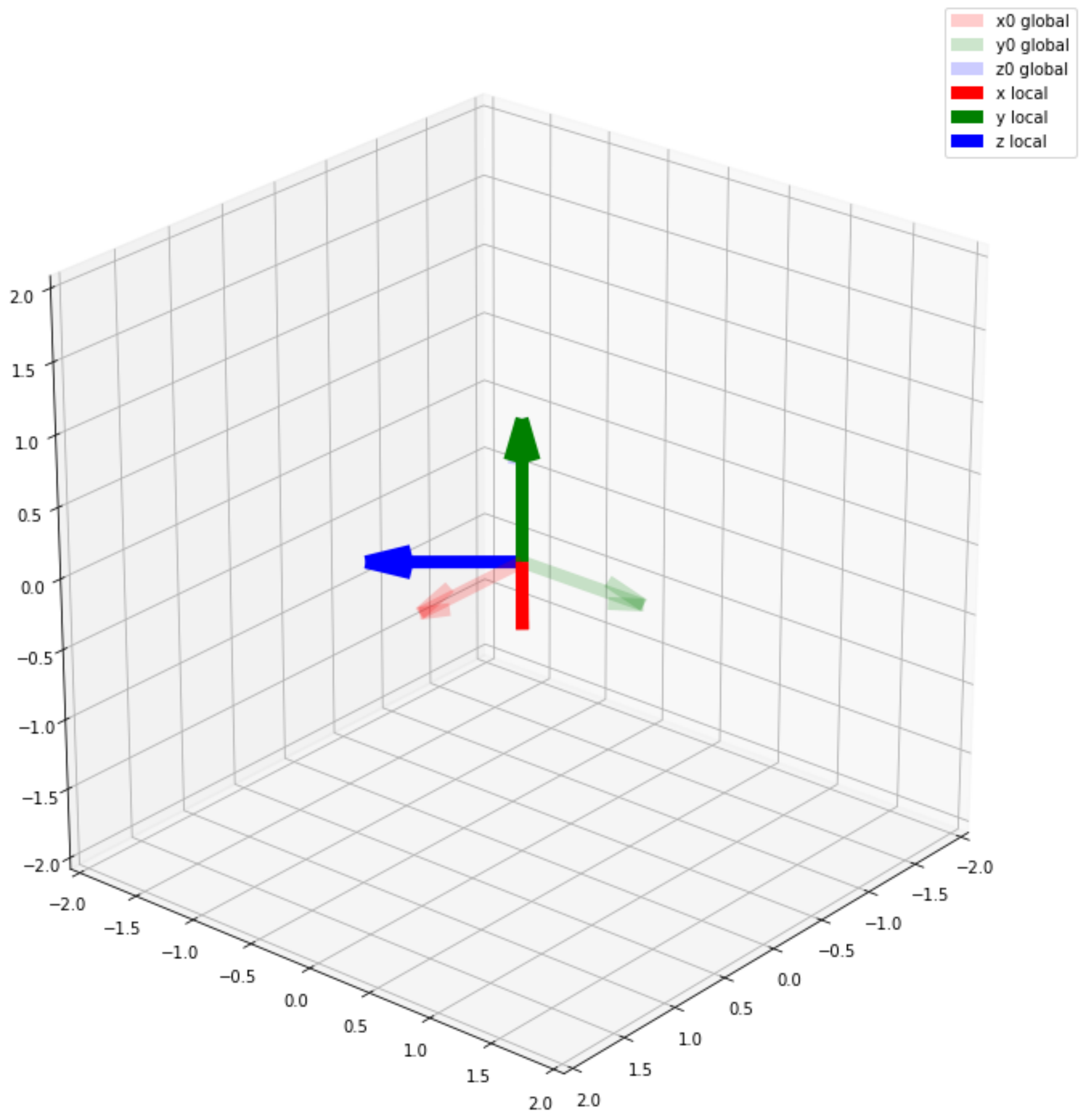
In this problem, you will use the function implemented above to draw the frames coincide with the given Euler angles.

```
def draw_vector(ax, origin, end, **kwargs):
  # ax.plot([origin[0], end[0]],[origin[1], end[1]], [origin[2], end[2]], **kwargs)
  ax.quiver(*origin, *(end-origin), **kwargs)


def draw_axes(ax=None, R=np.eye(3), offset = [0,0,0], draw_global_frame=True, **kwargs):
  """
  R: rotation matrix 3x3
  offset: origin offset 3x1
  """
  offset = np.array(offset)

  x0 = np.zeros(3)
  y0 = np.zeros(3)
  z0 = np.zeros(3)

  x0 += offset
  y0 += offset
  z0 += offset

  x0[0] += 1
  y0[1] += 1
  z0[2] += 1

  x_e = R[:, 0] + offset #x-axis arrow head
  y_e = R[:, 1] + offset
  z_e = R[:, 2] + offset

  if ax is None:
    fig= plt.figure(figsize=(10,10))
    ax = p3.Axes3D(fig)
  if draw_global_frame:
    draw_vector(ax, offset, x0, lw=8, color = 'r', label='x0 global', alpha=0.2)
    draw_vector(ax, offset, y0, lw=8, color = 'g', label='y0 global', alpha=0.2)
    draw_vector(ax, offset, z0, lw=8, color = 'b', label='z0 global', alpha=0.2)
  draw_vector(ax, offset, x_e, lw=8, color = 'r', label='x local')
  draw_vector(ax, offset, y_e, lw=8, color = 'g', label='y local')
  draw_vector(ax, offset, z_e, lw=8, color = 'b', label='z local')
  ax.set_xlim([-2, 2])
  ax.set_ylim([-2, 2])
  ax.set_zlim([-2, 2])

  return ax
```

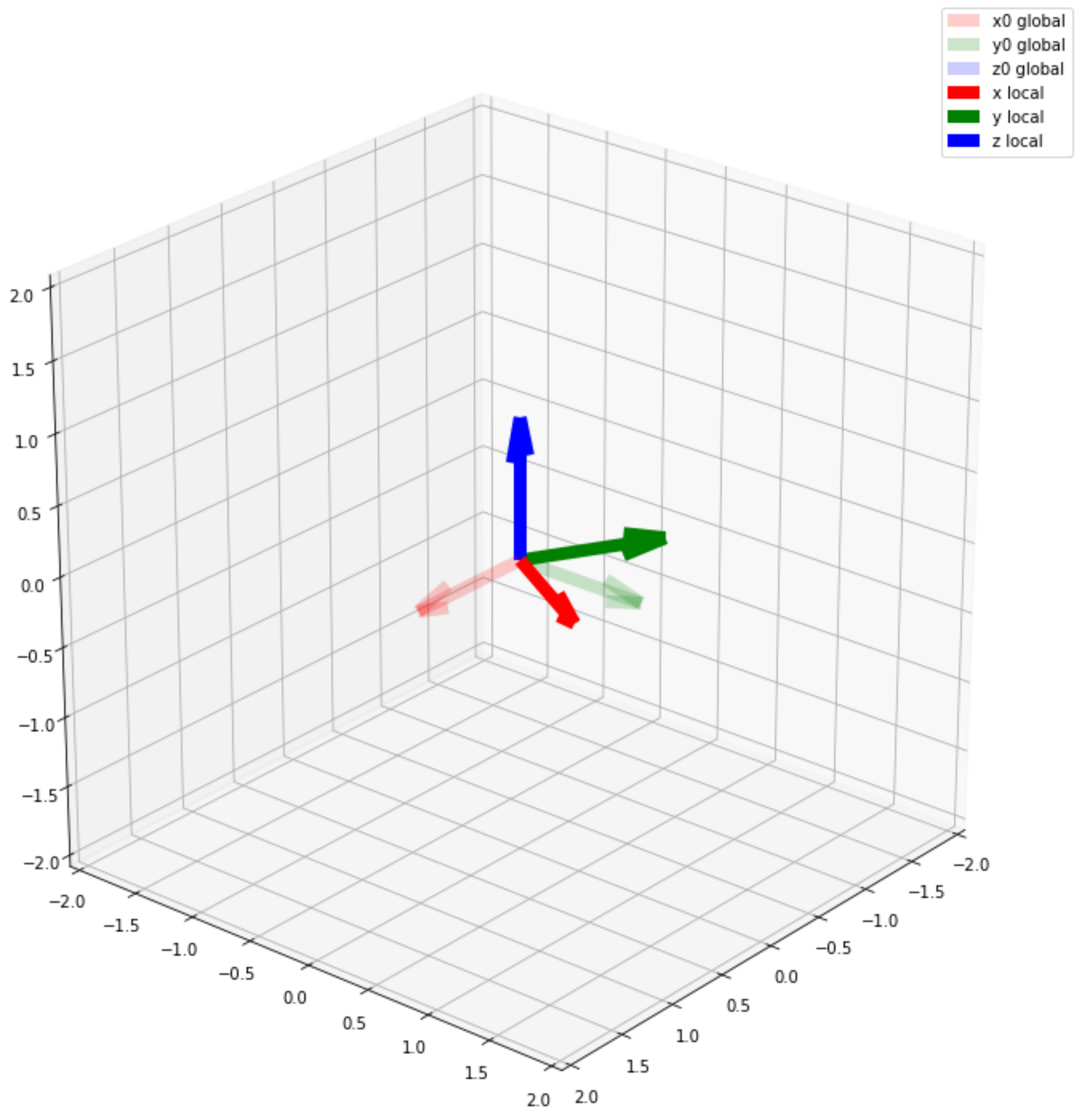
```
#(a)
R_a = ZYX_to_SO3(0.3, 0.2, 0.7) # Fill your code: Make a rotation matrix R_a

fig_a= plt.figure(figsize=(10,10))
ax_a = p3.Axes3D(fig_a)
ax_a = draw_axes(ax=ax_a, R=R_a)
ax_a.legend()
ax_a.view_init(25, 40)

fig_a.show()
```



```
#(b)
R_b = ZYX_to_SO3(0.7, 0, np.pi/2) # Fill your code

fig_b= plt.figure(figsize=(10,10))
ax_b= p3.Axes3D(fig_b)
ax_b = draw_axes(ax_b, R=R_b)
ax_b.view_init(25, 40)
ax_b.legend()

fig_b.show()
```



```
#(c)
R_c = ZYX_to_SO3(np.pi/3, 0, 0) # Fill your code

fig_c= plt.figure(figsize=(10,10))
ax_c = p3.Axes3D(fig_c)
ax_c = draw_axes(ax=ax_c, R=R_c)
ax_c.legend()
ax_c.view_init(25, 40)

fig_c.show()
```



Q.3 Make a function that converts ZYX Euler angle representation to Quaternion representation [10 pts]

```
def vee(R):
  return np.array([R[2,1], R[0, 2], R[1, 0]])
def SO3_to_so3(R):
  omega, theta = None, 0
  if np.allclose(R, np.eye(3)): # if R = identity
    pass
  elif np.abs(np.trace(R)+1)<1e-5: # if theta = pi
    theta = np.pi
    r13 = R[0,2]
    r23 = R[1,2]
    r33 = R[2,2]
    omega = np.array([r13,r23, 1+r33])
    omega *= (1./np.sqrt(2*(1+r33)))
  else: # Other normal cases
    # Fill your code to compute so(3)
    theta = np.arccos((np.trace(R) - 1) / 2)
    w = R - np.transpose(R)
    w *= 1 / (2 * np.sin(theta))
    omega = np.array([w[2,1], w[0,2], w[1,0]])

  return omega, theta


def ZYX_to_Quaternion(th_z, th_y, th_x):
  R= ZYX_to_SO3(th_z, th_y, th_x)
  omega, theta = SO3_to_so3(R)
  quaternion = np.zeros(4)
  if omega is None:
    quaternion[0] = 1.0
    return quaternion
  quaternion[0] = np.cos(theta/2.)
  quaternion[1] = np.sin(theta/2.)*omega[0]
  quaternion[2] = np.sin(theta/2.)*omega[1]
  quaternion[3] = np.sin(theta/2.)*omega[2]
  return quaternion


def test_ZYX_to_Quaternion():
  zyx_test = np.array([[0.3, 0.2, 0.7],
                       [0.7, np.pi, np.pi/2],
                       [np.pi/3, 0, 0]])

  soln = np.array([[ 0.92929998,  0.32333918,  0.14371374,  0.10582853],
                [ 0.24246536, -0.24246536,  0.66423682, -0.66423682],
                [ 0.8660254,   0.,          0.,          0.5       ]])
  res = []
  for test in zyx_test:
    res.append(ZYX_to_Quaternion(*test))
  res = np.array(res)
  if np.allclose(res, soln):
    print('your implementation is correct')
  else:
    print('In correct implementation try again')


test_ZYX_to_Quaternion()
```

```
your implementation is correct
```

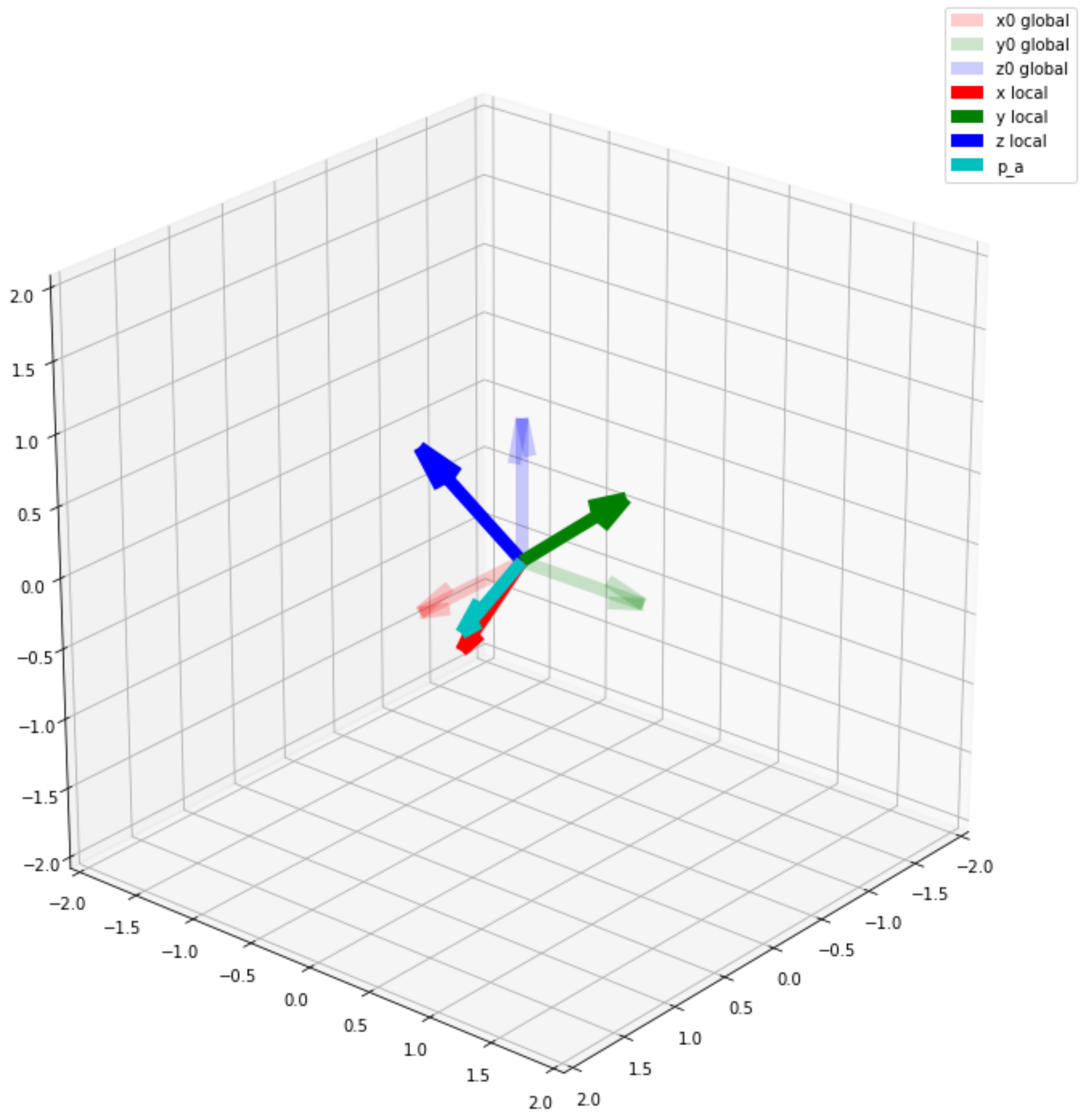
Q.4 Rotate the following vectors with the rotation matrices in Q.1 [10 pts]

$$p = (1, 0.1, 0.1)$$

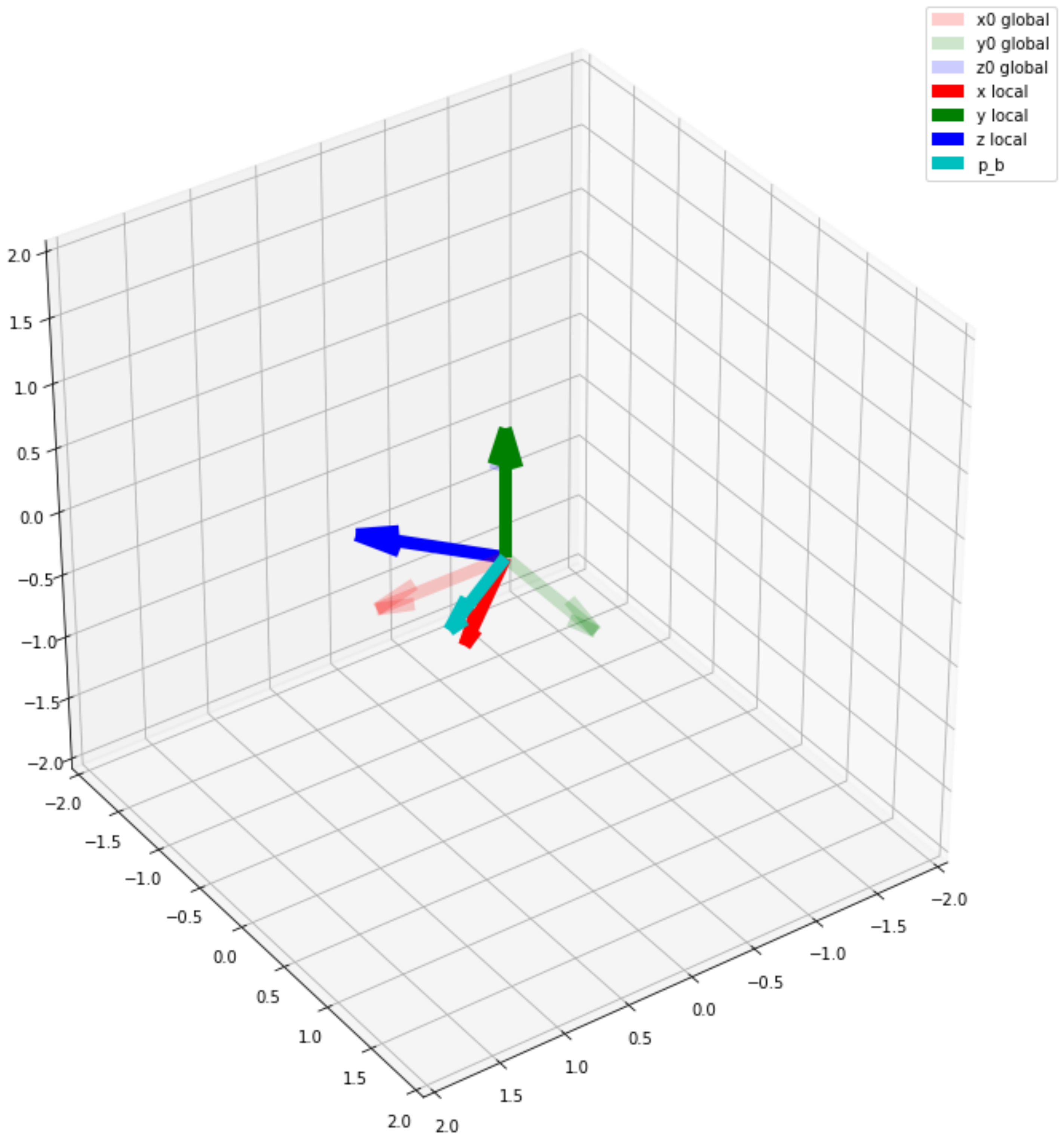
Visualize it and provide the coordinate of the rotated vector in both local and global coordinates.

```
p = np.array([1, 0.1, 0.1])
```

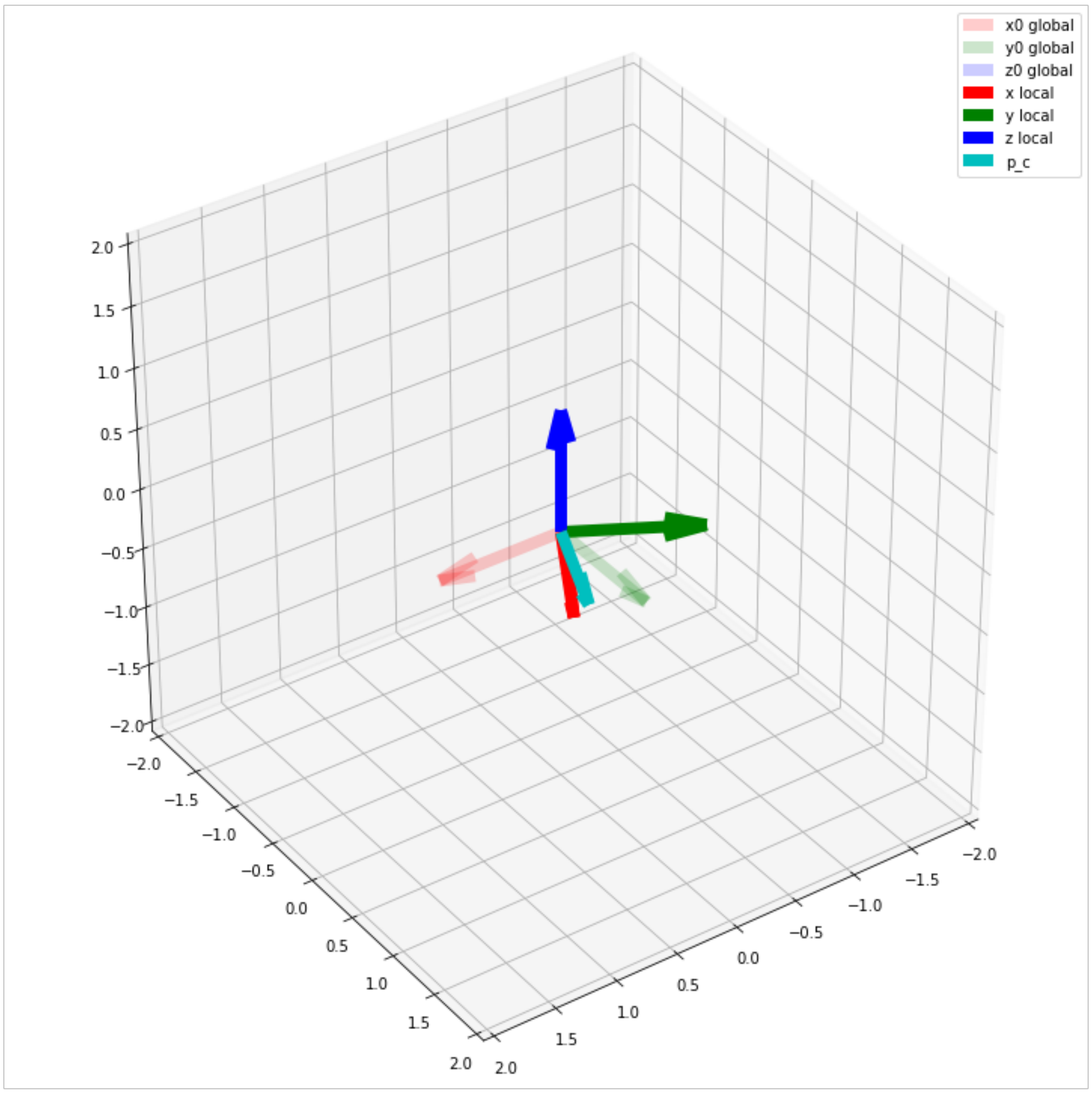
```
# Fill your code: Compute the vector (p_a) rotated by R_a
p_a = R_a @ p

fig_a= plt.figure(figsize=(10,10))
ax_a = p3.Axes3D(fig_a)
ax_a = draw_axes(ax=ax_a, R=R_a)
draw_vector(ax_a, [0, 0, 0], p_a, color='c', label='p_a', lw=8)
ax_a.legend()
ax_a.view_init(25, 40)

fig_a.show()
```



```
# Fill your code: Compute the vector (p_b) rotated by R_b
p_b = R_b @ p

fig_b= plt.figure(figsize=(10,10))
ax_b= p3.Axes3D(fig_b)
ax_b = draw_axes(ax=ax_b, R=R_b)
draw_vector(ax_b, [0, 0, 0], p_b, color='c', label='p_b', lw=8)
ax_b.legend()
ax_b.view_init(35, 55)


fig_b.show()
```



```
# Fill your code: Compute the vector (p_c) rotated by R_c
p_c = R_c @ p

fig_c= plt.figure(figsize=(10,10))
ax_c = p3.Axes3D(fig_c)
ax_c = draw_axes(ax=ax_c, R=R_c)
draw_vector(ax_c, [0, 0, 0], p_c, color='c', label='p_c', lw=8)
ax_c.legend()
ax_c.view_init(35, 55)

fig_c.show()
```



What is the coordinate of the rotated vectors in the local and global frames? Explain why the values read in the local frame do not change. Put your answers in the text box below.

Double-click (or enter) to edit

Q.5 (a) Integrate constant anglar velocity of $\omega = [10, 0, 0]\ [rad/s]$ for 1.5 sec starting from the following orientation [10 pts]

$$R = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Q.5 (b) Animate and explain the difference of explicit and implicit integration [10pts]

Hint: for matrix exponential operation use scipy.linalg.expm function

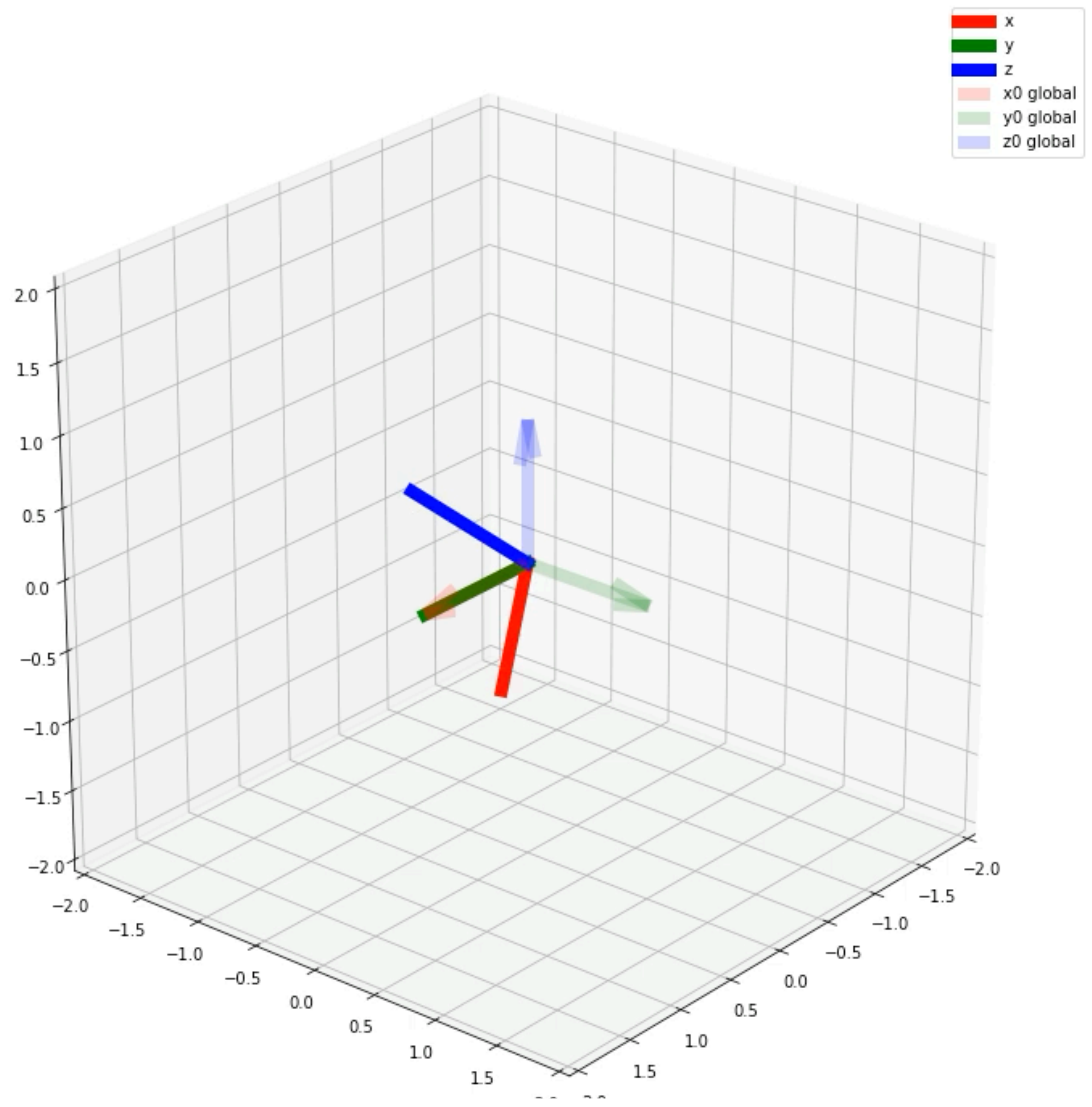
```
def hat(a):
  return np.array([[0, -a[2], a[1]],
                   [a[2], 0, -a[0]],
                   [-a[1], a[0], 0]])
```

```
def draw_axes_mp(Rs, **kwargs):
  """
  Rs: a list of rotation matrices
  """
  fig= plt.figure(figsize=(10,10))
  ax = p3.Axes3D(fig)
  origin = np.zeros(3)

  x0 = np.zeros(3)
  y0 = np.zeros(3)
  z0 = np.zeros(3)

  x0[0] += 1
  y0[1] += 1
  z0[2] += 1

  draw_vector(ax, origin, x0, lw=8, color = 'r', label='x0 global', alpha=0.2)
  draw_vector(ax, origin, y0, lw=8, color = 'g', label='y0 global', alpha=0.2)
  draw_vector(ax, origin, z0, lw=8, color = 'b', label='z0 global', alpha=0.2)

  x_axis,  = ax.plot([0, 1], [0, 0], [0, 0], lw=8, color='r', label='x')
  y_axis,  = ax.plot([0, 0], [0, 1], [0, 0], lw=8, color='g', label='y')
  z_axis,  = ax.plot([0, 0], [0, 0], [0, 1], lw=8, color='b', label='z')

  ax.set_xlim([-2, 2])
  ax.set_ylim([-2, 2])
  ax.set_zlim([-2, 2])
  ax.legend()
  ax.view_init(25, 40)


  def drawFrame(k):
    x_e = Rs[k][:, 0]
    y_e = Rs[k][:, 1]
    z_e = Rs[k][:, 2]
    x_axis.set_data([0, x_e[0]], [0, x_e[1]])
    x_axis.set_3d_properties([0, x_e[2]])
    y_axis.set_data([0, y_e[0]], [0, y_e[1]])
    y_axis.set_3d_properties([0, y_e[2]])
    z_axis.set_data([0, z_e[0]], [0, z_e[1]])
    z_axis.set_3d_properties([0, z_e[2]])

    # return x0_axis, y0_axis, z0_axis, x_axis, y_axis, z_axis,
    return x_axis, y_axis, z_axis,

  if 'frames' in kwargs:
    anim = animation.FuncAnimation(fig, drawFrame, frames=kwargs['frames'], interval=1000, blit=True)
  else:
    anim = animation.FuncAnimation(fig, drawFrame, frames=10, interval=1000, blit=True)
  return anim
```

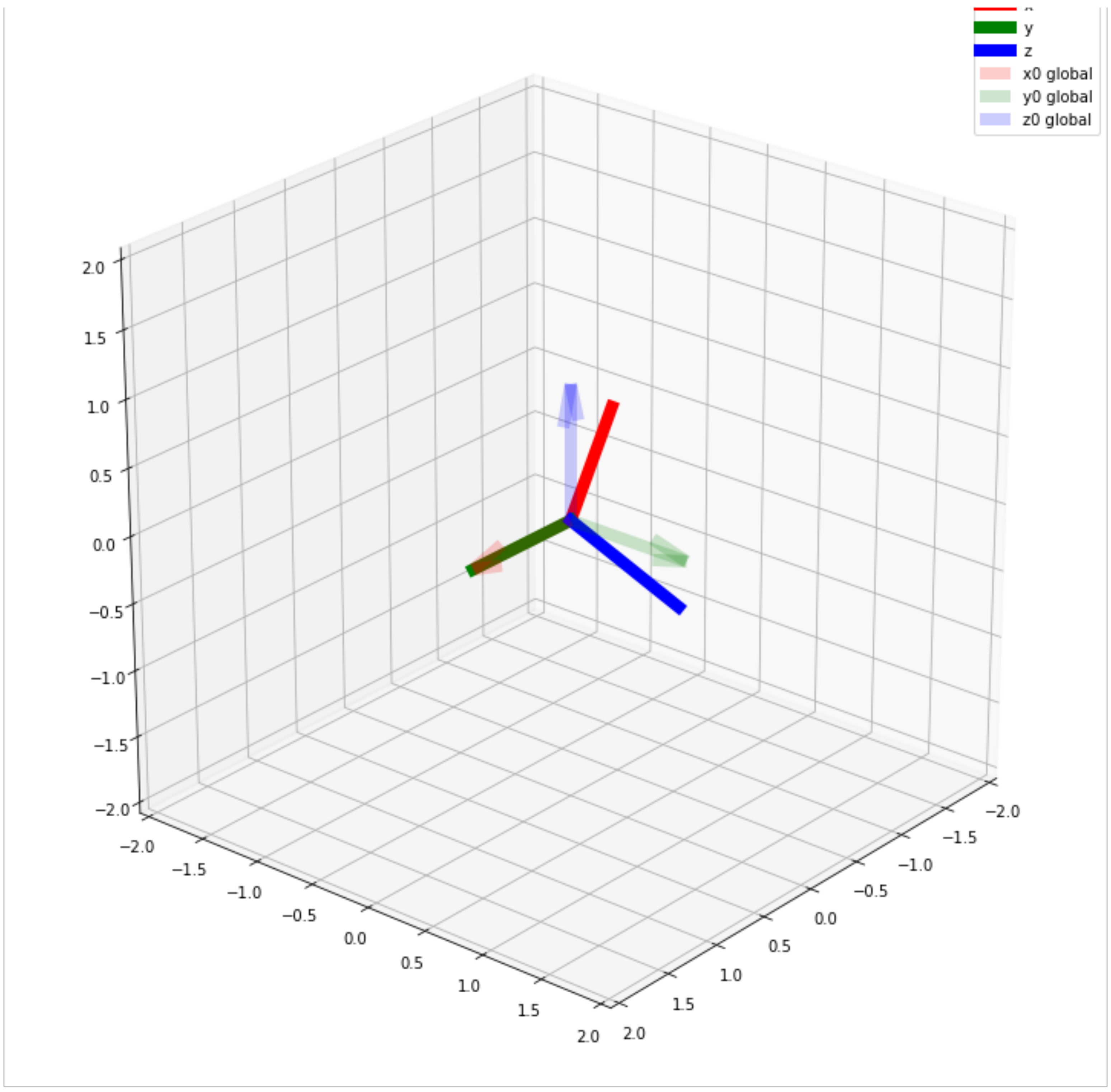
```
omega = np.array([10, 0, 0])
N_t = 30
delta_t = 0.015
R = np.array([[0, 1, 0],
              [-1, 0, 0],
              [0, 0, 1]])

omega_t = omega*delta_t
R_implicit = np.zeros((N_t, 3, 3))
R_explicit = np.zeros((N_t, 3, 3))

R_implicit[0, :, :] = R
R_explicit[0, :, :] = R

exp_accumulated = np.eye(3)

omega_hat = hat(omega_t)

for i in range(N_t-1):

  e_o_t_hat = scipy.linalg.expm(omega_hat)

  R_implicit[i+1, :, :] = R_implicit[i,:,:] @ e_o_t_hat # Fill your code
  R_explicit[i+1, :, :] = e_o_t_hat @ R_explicit[i,:,:] # Fill your code
```

Animate explicit rotation

```
anim = draw_axes_mp(R_explicit, frames=N_t)
HTML(anim.to_html5_video())
```

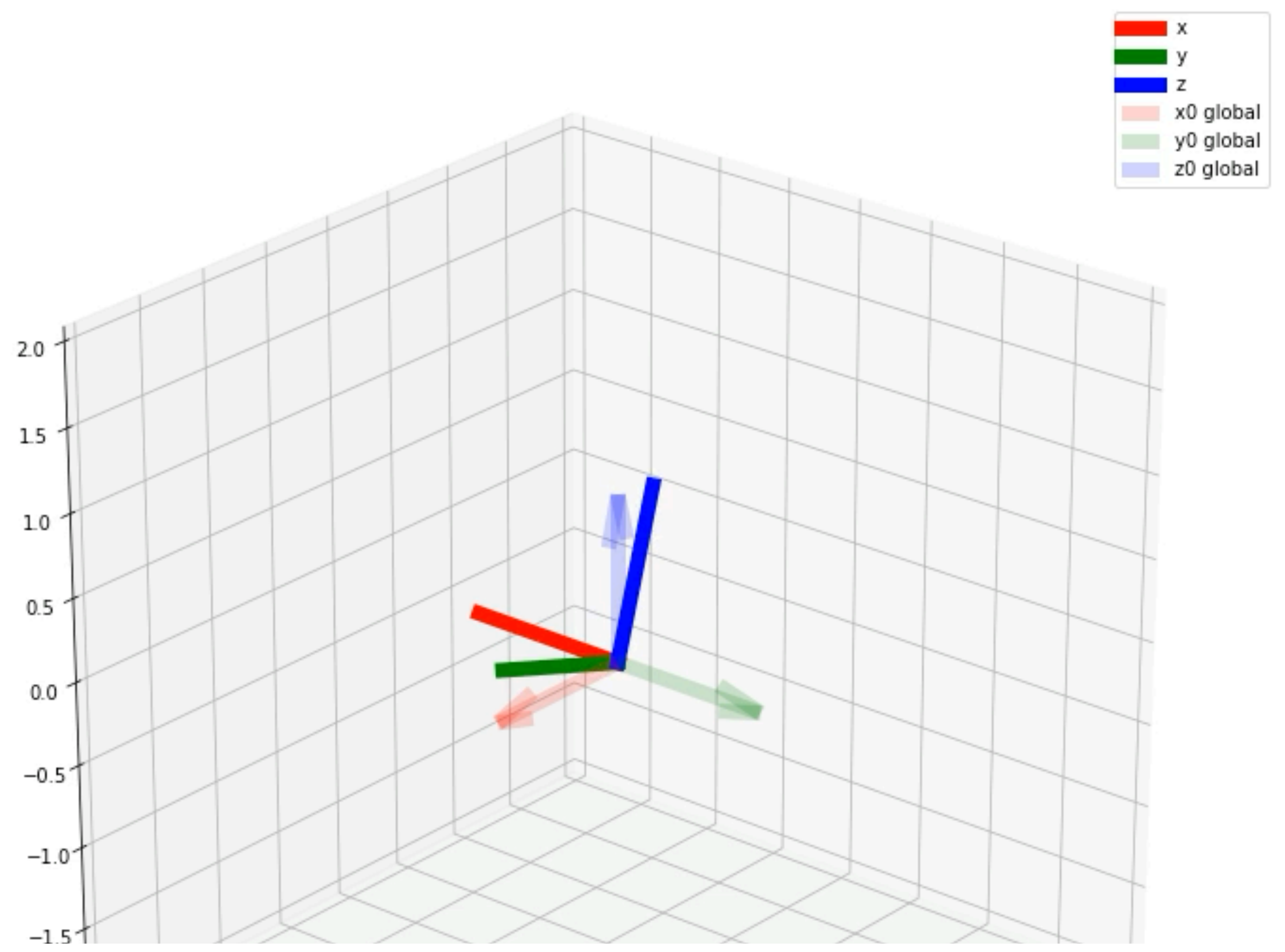


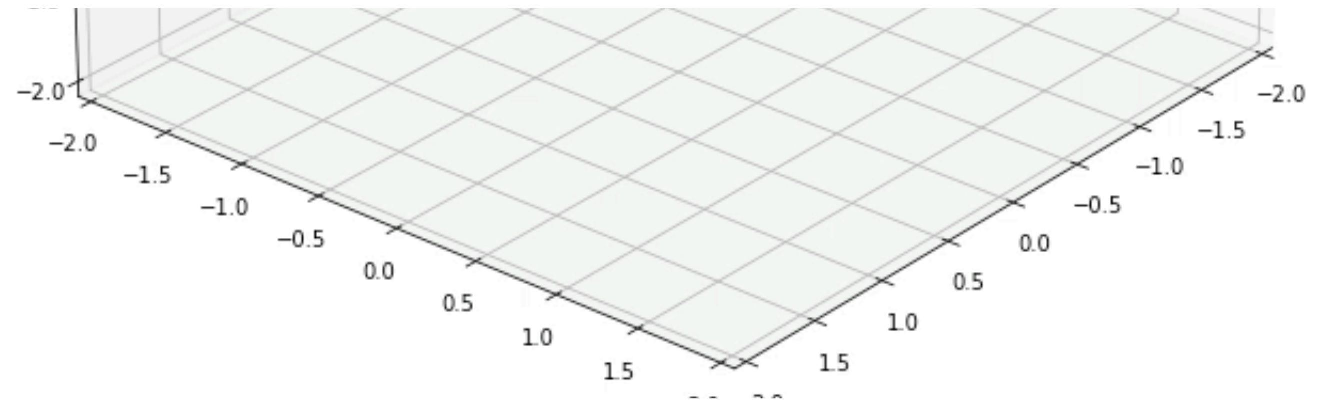
0:09 0:30



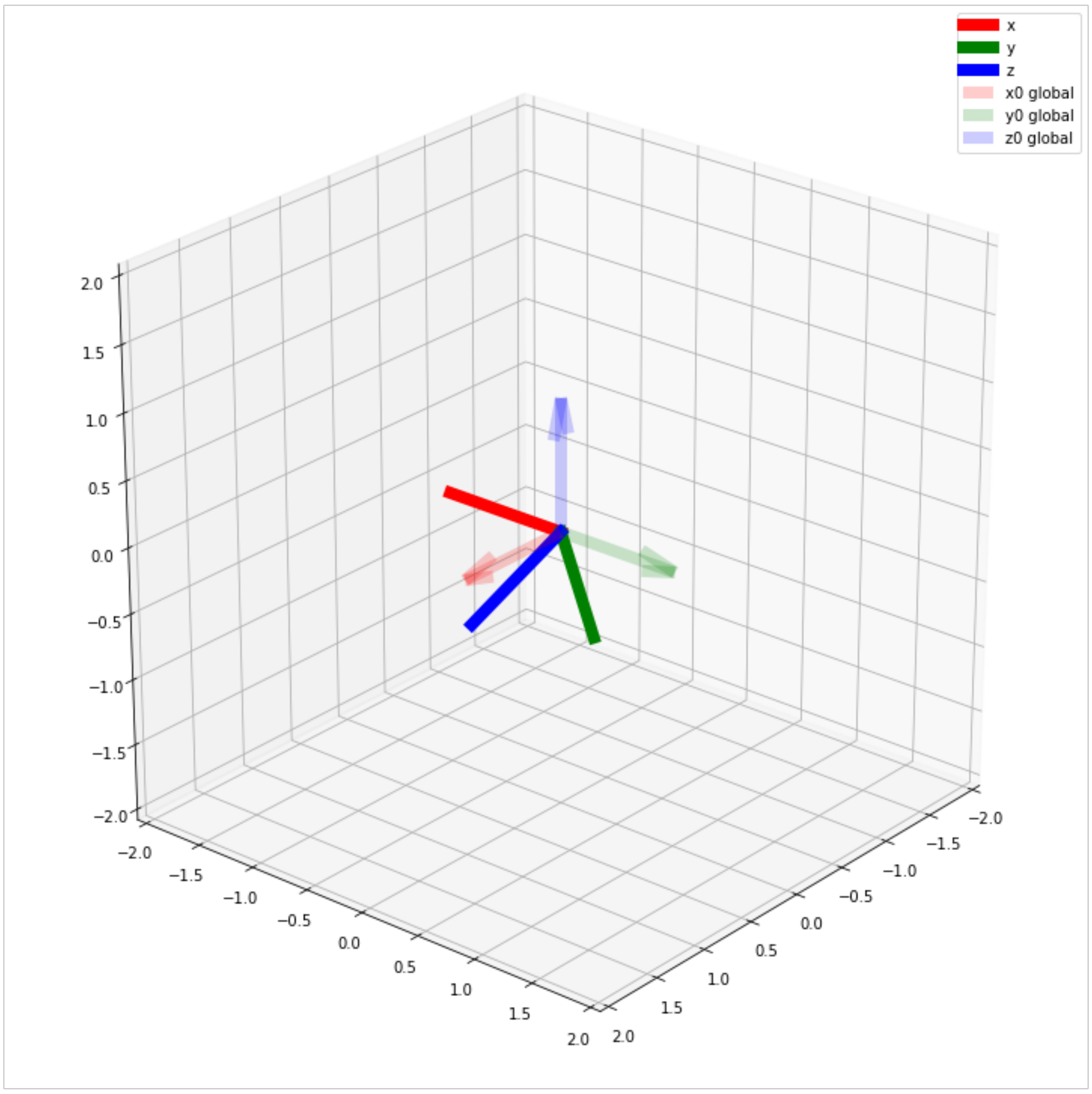
Animate implicit rotation

```
anim = draw_axes_mp(R_implicit, frames=N_t)
HTML(anim.to_html5_video())
```





0:02 0:30



Explain the difference of implicit and explicit rotation based on the integration results

Implicit is rotating the axis with respect to itself while explicit is rotating with respect to the global xyz.

Q.6 Make a function that converts position and ZYX orientation representation to SE(3) [10 pts].

```
def POS_ZYX_to_SE3(x, y, z, th_z, th_y, th_x):
  SE3  = np.eye(4)
  # Fill your code
  R = ZYX_to_SO3(th_z, th_y, th_x)
  SE3[0:3, 0:3] = R
  SE3[0,3] = x
  SE3[1,3] = y
  SE3[2,3] = z

  return SE3


def test_POS_ZYX_to_SE3():
  zyx_test = np.array([[0.3, 0.2, 0.7, 0.5, 0.2, 1.2],
                     [0.5, 0.6, 1.8, 0.7, np.pi, np.pi/2],
                     [0.1, 0.0, 2, np.pi/3, 0, 0]])
  soln = np.array([[[ 8.60089338e-01, -1.12237211e-02,  5.10019959e-01,  3.00000000e-01],
                    [ 4.69868947e-01,  4.06772914e-01, -7.83427705e-01,  2.00000000e-01],
                    [-1.98669331e-01,  9.13460357e-01,  3.55134724e-01,  7.00000000e-01],
                    [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  1.00000000e+00]],
                   [[-7.64842187e-01,  5.42191972e-17,  6.44217687e-01,  5.00000000e-01],
                    [-6.44217687e-01,  1.25726990e-16, -7.64842187e-01,  6.00000000e-01],
                    [-1.22464680e-16, -1.00000000e+00, -6.12323400e-17,  1.80000000e+00],
                    [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  1.00000000e+00]],
                   [[ 5.00000000e-01, -8.66025404e-01,  0.00000000e+00,  1.00000000e-01],
                    [ 8.66025404e-01,  5.00000000e-01,  0.00000000e+00,  0.00000000e+00],
                    [ 0.00000000e+00,  0.00000000e+00,  1.00000000e+00,  2.00000000e+00],
                    [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  1.00000000e+00]]])
  res = []
  for test in zyx_test:
    res.append(POS_ZYX_to_SE3(*test))
  res = np.array(res)
  if np.allclose(res, soln):
    print('your implementation is correct')
  else:
    print('In correct implementation try again')


test_POS_ZYX_to_SE3()
```

```
    your implementation is correct
```

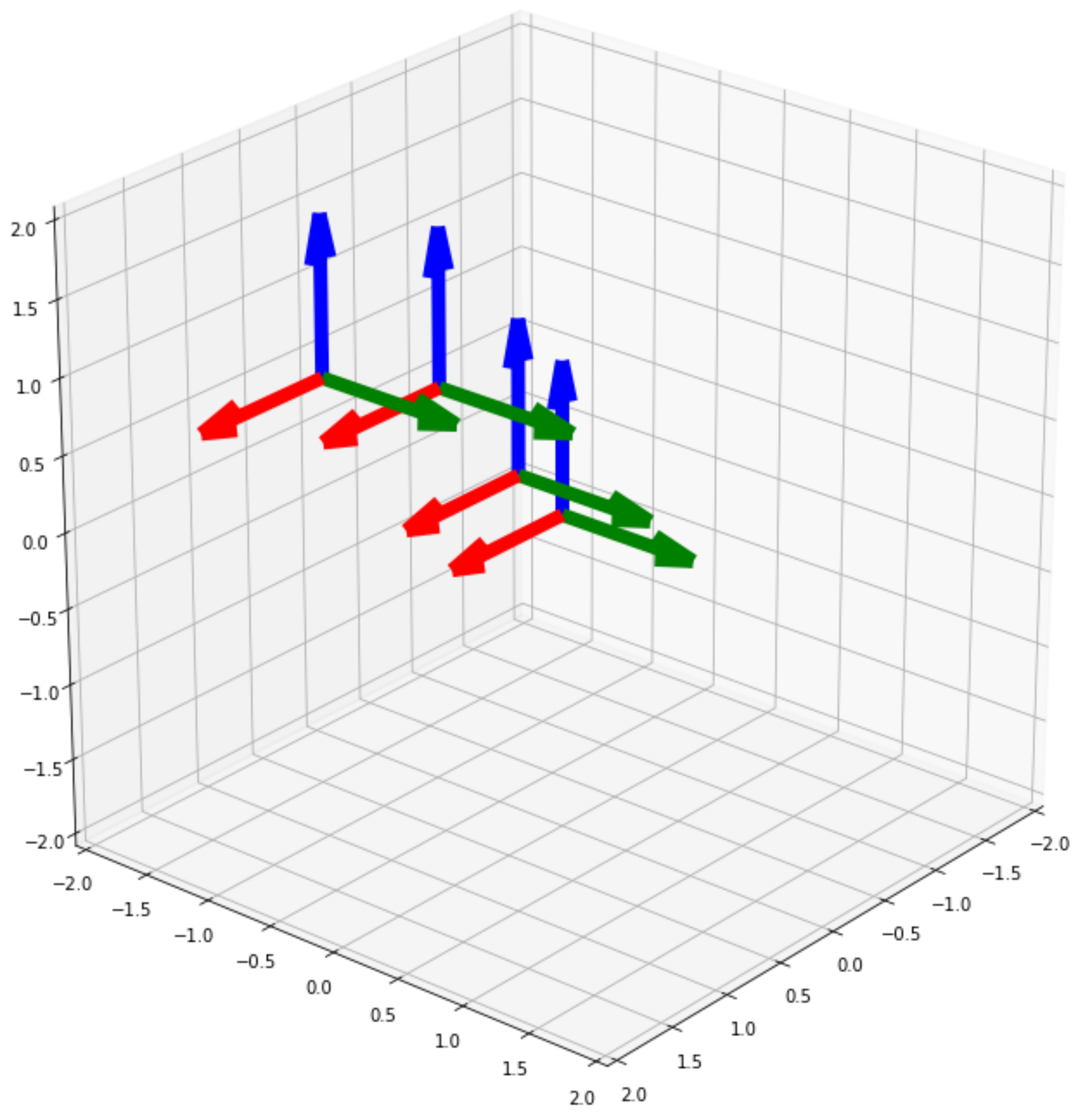
Q.7 Using the following relationships between frames, find the SE(3) that corresponds to $frame\{i\} \rightarrow \{0\} \; \forall \; i \in \{0, 1, 2, 3\}$ and draw them on the same plot [20 pts].

> > - frame$\{0\}$: $position = (0, 0, 0), \; ZYX = (0, 0, 0)$
> > - frame$\{0\} \rightarrow \{1\}$: $position = (0.4, 0.0, 0.4), \; ZYX = (0, \theta_1, 0)$
> > - frame$\{1\} \rightarrow \{2\}$: $position = (0.7, 0.0, 0.8), \; ZYX = (0, \theta_2, 0)$
> > - frame$\{2\} \rightarrow \{3\}$: $position = (1.0, 0.0, 0.4), \; ZYX = (0, \theta_3, 0)$

For this question use $\theta_1 = \theta_2 = \theta_3 = 0$

```
def draw_SE3(ax, T):
  ax = draw_axes(ax, R=T[:3, :3], offset = T[:-1, 3], draw_global_frame=False)
  return ax
```

```
T0 = POS_ZYX_to_SE3(-0.0, 0.0, 0.0, 0., 0.0, 0.0)

# Fill your code: Complete all SE(3)
T0_1 = POS_ZYX_to_SE3(.4, 0.0, .4, 0., 0.0, 0.0)
T1_2 = POS_ZYX_to_SE3(.7, 0.0, .8, 0., 0.0, 0.0)
T2_3 = POS_ZYX_to_SE3(1.0, 0.0, .4, 0., 0.0, 0.0)
T0_2 = T0_1 @ T1_2
T0_3 = T0_1 @ T1_2 @ T2_3


fig_T1= plt.figure(figsize=(10,10))
ax = p3.Axes3D(fig_T1)
ax.view_init(25, 40)

ax = draw_SE3(ax, T0)
ax = draw_SE3(ax, T0_1)
ax = draw_SE3(ax, T0_2)
ax = draw_SE3(ax, T0_3)
```



Q.8 In this question, the pitch angles in Q.7 change over time using the following rule [10 pts]

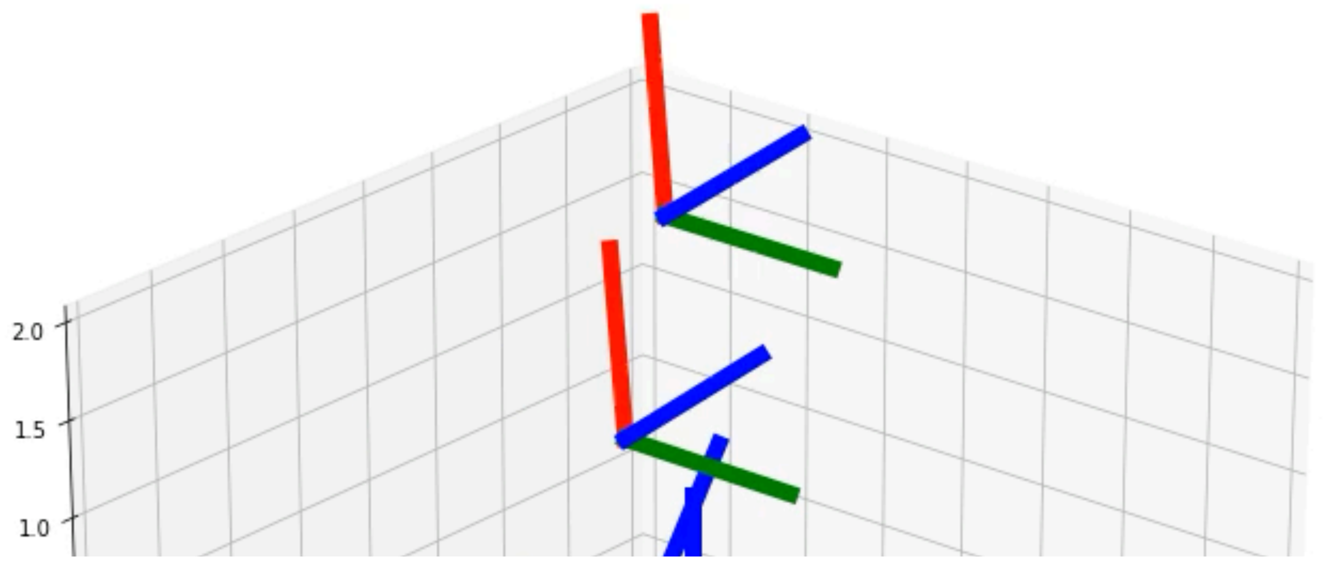
$$\theta_1(t) = sin(\omega t)$$
$$\theta_2(t) = sin(\omega t + 0.3)$$
$$\theta_3(t) = 0.$$

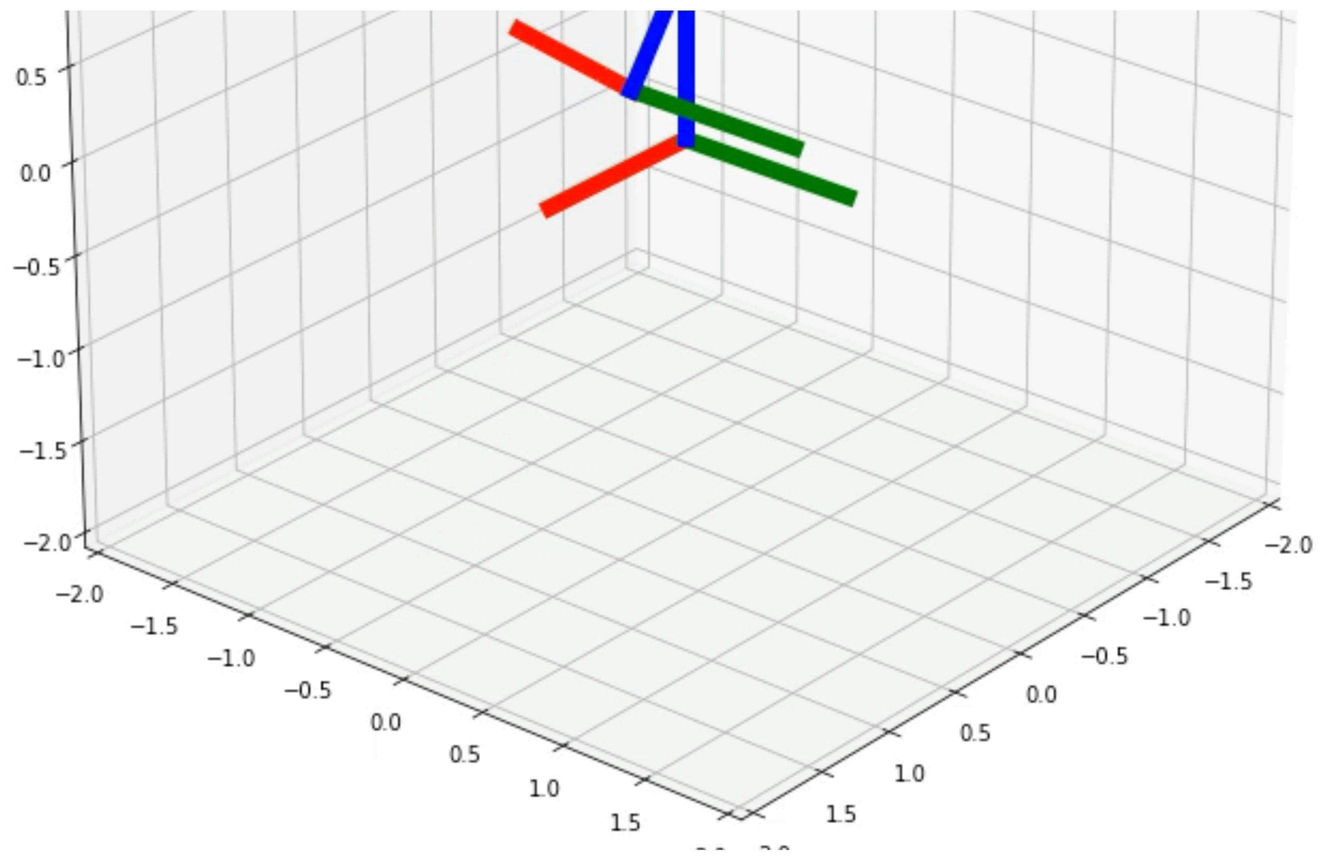
where $\omega = 5\ [rad/s]$. Animate the frames $t \in [0, 1.5]$ sec.

```
N = 50
omega = 5
t = np.linspace(0, 1.5, N)
th_1_hist  = np.sin(omega*t)
th_2_hist  = np.sin(omega*t + 0.3)
th_3_hist  = 0*np.sin(omega*t)
T_hist = np.zeros((N, 4, 4, 4))
for i in range(N):
  T0 = POS_ZYX_to_SE3(-0.0, 0.0, 0.0, 0., 0.0, 0.0)

  # Fill your code: complete SE(3)
  T0_1 = POS_ZYX_to_SE3(.4, 0.0, .4, 0., th_1_hist[i], 0.0)
  T1_2 = POS_ZYX_to_SE3(.7, 0.0, .8, 0., th_2_hist[i], 0.0)
  T2_3 = POS_ZYX_to_SE3(1.0, 0.0, .4, 0., th_3_hist[i], 0.0)
  T0_2 = T0_1 @ T1_2
  T0_3 = T0_1 @ T1_2 @ T2_3

  T_hist[i, 0, :, :] = T0
  T_hist[i, 1, :, :] = T0_1
  T_hist[i, 2, :, :] = T0_2
  T_hist[i, 3, :, :] = T0_3
```

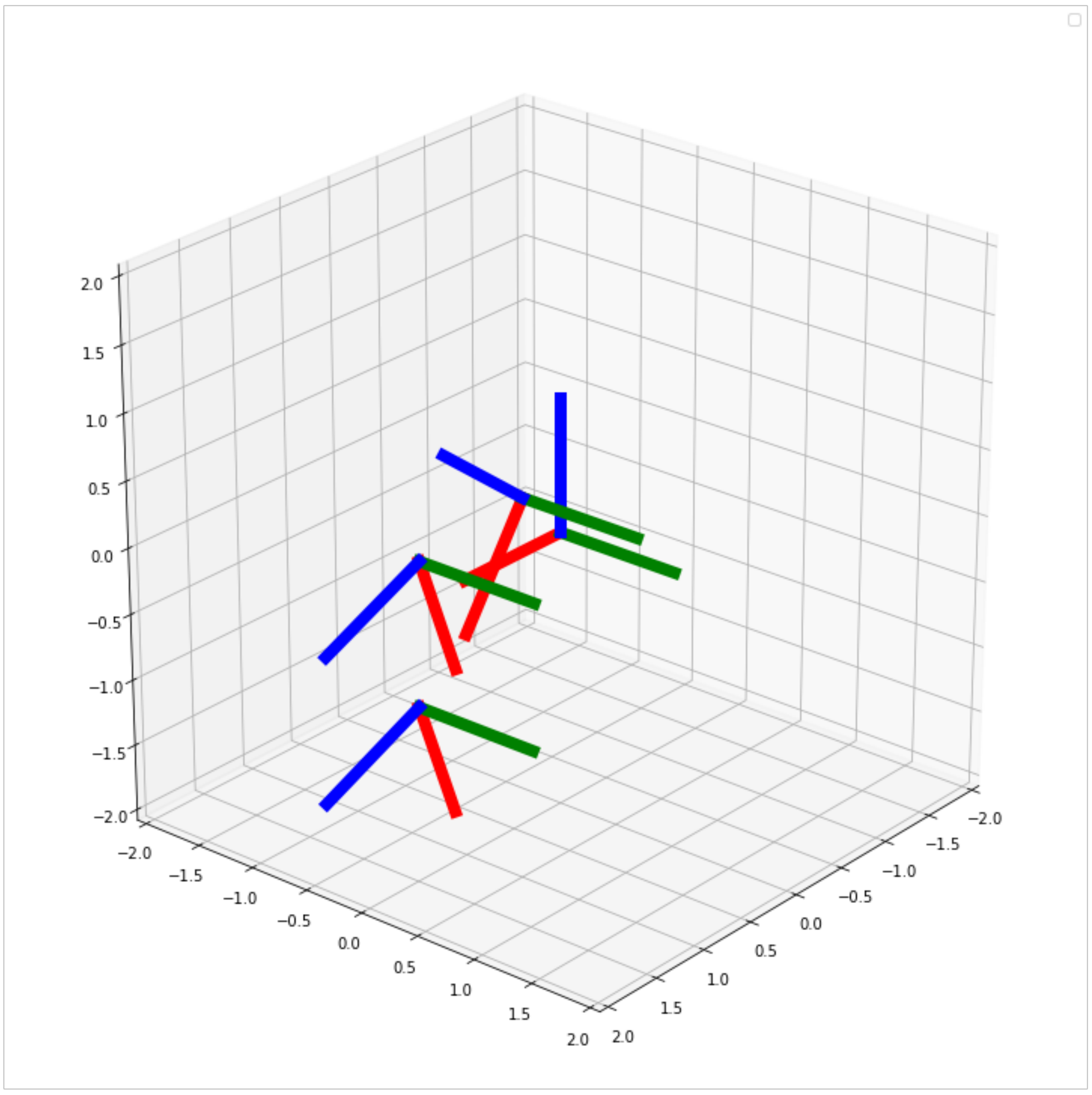
```
def animate_frames(T_hist, **kwargs):
  """
  T_hist: a list of SE(3) matrices (N, n_frames, 4, 4)
  """
  fig= plt.figure(figsize=(10,10))
  ax = p3.Axes3D(fig)

  N, n_frames, _, _ = T_hist.shape
  x_axis_list = []
  y_axis_list = []
  z_axis_list = []

  for i in range(n_frames):
    x_axis,  = ax.plot([0, 1], [0, 0], [0, 0], lw=8, color='r')
    y_axis,  = ax.plot([0, 0], [0, 1], [0, 0], lw=8, color='g')
    z_axis,  = ax.plot([0, 0], [0, 0], [0, 1], lw=8, color='b')
    x_axis_list.append(x_axis)
    y_axis_list.append(y_axis)
    z_axis_list.append(z_axis)

  ax.set_xlim([-2, 2])
  ax.set_ylim([-2, 2])
  ax.set_zlim([-2, 2])
  ax.legend()
  ax.view_init(25, 40)


  def drawFrame(k):
    for i in range(n_frames):
      T = T_hist[k, i, :, :]
      R=T[:3, :3]
      offset = T[:-1, 3]
      x_e = R[:, 0] + offset
      y_e = R[:, 1] + offset
      z_e = R[:, 2] + offset

      x_axis_list[i].set_data([offset[0], x_e[0]], [offset[1], x_e[1]])
      x_axis_list[i].set_3d_properties([offset[2], x_e[2]])
      y_axis_list[i].set_data([offset[0], y_e[0]], [offset[1], y_e[1]])
      y_axis_list[i].set_3d_properties([offset[2], y_e[2]])
      z_axis_list[i].set_data([offset[0], z_e[0]], [offset[1], z_e[1]])
      z_axis_list[i].set_3d_properties([offset[2], z_e[2]])
    return x_axis_list + y_axis_list + z_axis_list

  if 'frames' in kwargs:
    anim = animation.FuncAnimation(fig, drawFrame, frames=kwargs['frames'], interval=1000, blit=True)
  else:
    anim = animation.FuncAnimation(fig, drawFrame, frames=10, interval=1000, blit=True)
  return anim



anim = animate_frames(T_hist, frames=N)
HTML(anim.to_html5_video())
```

```
    WARNING:matplotlib.legend:No handles with labels found to put in legend.
```




0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

0:25 0:50


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0