3D-Point

April 3, 2025

1 3D Point Moving in a Straight Line

Arthur Ryman, last updated 2025-04-03

1.1 Introduction

The goal of this notebook is to understand the 3D animation capabilities of matplotlib. I will attempt to animate a point moving in a straight line.

The following code was generated by ChatGPT in response to a series of prompts.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from mpl_toolkits.mplot3d import Axes3D
from IPython.display import HTML

# Initialize the figure and 3D axis
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Generate the initial data
t = np.linspace(0, 2*np.pi, 100)
x = np.sin(t)
y = np.cos(t)
z = t
line, = ax.plot(x, y, z, lw=2)

# Set limits and labels
ax.set_xlim([-1, 1])
ax.set_ylim([-1, 1])
ax.set_zlim([0, 2*np.pi])
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')

# Animation function
def update(frame):
    z_new = t + frame * 0.1  # Move upwards over time
```

```
    line.set_data(np.sin(z_new), np.cos(z_new))
    line.set_3d_properties(z_new)  # Update z-data
    return line,

# Create and store animation
ani = animation.FuncAnimation(fig, update, frames=100, interval=50, blit=False)

# Suppress extra static plot
plt.close(fig)

# Display animation in Jupyter Notebook
HTML(ani.to_jshtml())
```

[1]: <IPython.core.display.HTML object>

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from mpl_toolkits.mplot3d import Axes3D
from IPython.display import HTML

# Initialize the figure and 3D axis
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Set limits and labels
ax.set_xlim([-1.5, 1.5])
ax.set_ylim([-1.5, 1.5])
ax.set_zlim([-1.5, 1.5])
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')

# Define start and end points
start = np.array([-1, -1, -1])
end = np.array([1, 1, 1])
num_frames = 100  # Number of frames in the animation

# Compute linearly spaced points for movement
points = np.linspace(start, end, num_frames)

# Function to create a sphere
def create_sphere(center, radius=0.1, resolution=20):
    u = np.linspace(0, 2 * np.pi, resolution)
    v = np.linspace(0, np.pi, resolution)
    x = radius * np.outer(np.cos(u), np.sin(v)) + center[0]
    y = radius * np.outer(np.sin(u), np.sin(v)) + center[1]
```

```python
    z = radius * np.outer(np.ones(np.size(u)), np.cos(v)) + center[2]
    return x, y, z

# Initialize sphere
x_sphere, y_sphere, z_sphere = create_sphere(start)
sphere = ax.plot_surface(x_sphere, y_sphere, z_sphere, color='r')

# Animation function
def update(frame):
    global sphere
    # Remove previous sphere
    for artist in ax.collections:
        artist.remove()

    # Compute new sphere position
    x_sphere, y_sphere, z_sphere = create_sphere(points[frame])

    # Draw new sphere
    sphere = ax.plot_surface(x_sphere, y_sphere, z_sphere, color='r')

    return sphere,

# Create animation
ani = animation.FuncAnimation(fig, update, frames=num_frames, interval=50,␣
↪blit=False)

# Suppress static plot
plt.close(fig)

# Display animation in Jupyter Notebook
HTML(ani.to_jshtml())
```

[2]: <IPython.core.display.HTML object>

[3]:
```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from mpl_toolkits.mplot3d import Axes3D, art3d
from IPython.display import HTML

# Define the icosahedron's vertices and faces
def create_icosahedron(radius=0.1):
    phi = (1 + np.sqrt(5)) / 2  # Golden ratio
    vertices = np.array([
        (-1,  phi, 0), (1,  phi, 0), (-1, -phi, 0), (1, -phi, 0),
        (0, -1,  phi), (0, 1,  phi), (0, -1, -phi), (0, 1, -phi),
        (phi, 0, -1), (phi, 0, 1), (-phi, 0, -1), (-phi, 0, 1)
```

```
    ])
    vertices /= np.linalg.norm(vertices[0])  # Normalize to unit sphere
    vertices *= radius  # Scale to desired radius

    faces = [
        (0, 11, 5), (0, 5, 1), (0, 1, 7), (0, 7, 10), (0, 10, 11),
        (1, 5, 9), (5, 11, 4), (11, 10, 2), (10, 7, 6), (7, 1, 8),
        (3, 9, 4), (3, 4, 2), (3, 2, 6), (3, 6, 8), (3, 8, 9),
        (4, 9, 5), (2, 4, 11), (6, 2, 10), (8, 6, 7), (9, 8, 1)
    ]

    return np.array(vertices), faces

# Generate the icosahedron
icosahedron_vertices, icosahedron_faces = create_icosahedron(radius=.05)

# Define the figure and 3D axis
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Set plot limits
ax.set_xlim([-1.5, 1.5])
ax.set_ylim([-1.5, 1.5])
ax.set_zlim([-1.5, 1.5])
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')

# Define start and end points for motion
start = np.array([-1, -1, -1])
end = np.array([1, 1, 1])
num_frames = 100

# Compute evenly spaced positions along the straight-line path
positions = np.linspace(start, end, num_frames)

# Initialize icosahedron poly collection
icosahedron_poly = art3d.Poly3DCollection([], color='r')

# Add to the axis
ax.add_collection3d(icosahedron_poly)

# Animation update function
def update(frame):
    # Compute the new position
    position = positions[frame]
```

```python
    # Translate icosahedron vertices to the new position
    transformed_vertices = icosahedron_vertices + position

    # Create the polygon faces
    poly_faces = [[transformed_vertices[i] for i in face] for face in␣
 ↪icosahedron_faces]

    # Update the polygon collection
    icosahedron_poly.set_verts(poly_faces)

    return icosahedron_poly,

# Create animation
ani = animation.FuncAnimation(fig, update, frames=num_frames, interval=50,␣
 ↪blit=False)

# Suppress static plot
plt.close(fig)

# Display animation in Jupyter Notebook
HTML(ani.to_jshtml())
```

[3]: <IPython.core.display.HTML object>

[4]:
```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
from IPython.display import HTML
import random

# Define cube vertices relative to the origin
def create_cube(size=0.2):
    half = size / 2
    vertices = np.array([
        [-half, -half, -half], [half, -half, -half], [half, half, -half],␣
 ↪[-half, half, -half],  # Bottom face
        [-half, -half, half], [half, -half, half], [half, half, half], [-half,␣
 ↪half, half]        # Top face
    ])

    faces = [
        [0, 1, 2, 3],  # Bottom face
        [4, 5, 6, 7],  # Top face
        [0, 1, 5, 4],  # Front face
        [2, 3, 7, 6],  # Back face
        [1, 2, 6, 5],  # Right face
```

```
        [4, 7, 3, 0],  # Left face
    ]

    return np.array(vertices), faces

# Generate cube
cube_vertices, cube_faces = create_cube(size=0.5)

# Define colors: Assign a random color to each face
color_choices = ['red', 'green', 'blue', 'white']
cube_colors = [random.choice(color_choices) for _ in range(6)]

# Create figure and 3D axis
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Set limits
ax.set_xlim([-1.5, 1.5])
ax.set_ylim([-1.5, 1.5])
ax.set_zlim([-1.5, 1.5])
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')

# Define start and end points
start = np.array([-1, -1, -1])
end = np.array([1, 1, 1])
num_frames = 100

# Compute evenly spaced positions for the cube's motion
positions = np.linspace(start, end, num_frames)

# Initialize cube poly collection
cube_poly = Poly3DCollection([], facecolors=cube_colors, edgecolor='black')

# Add cube to the axis
ax.add_collection3d(cube_poly)

# Animation update function
def update(frame):
    # Compute the new position
    position = positions[frame]

    # Translate cube vertices to the new position
    transformed_vertices = cube_vertices + position

    # Create updated cube faces
```

```
    poly_faces = [[transformed_vertices[i] for i in face] for face in␣
↪cube_faces]

    # Update the cube polygon collection
    cube_poly.set_verts(poly_faces)

    return cube_poly,

# Create animation
ani = animation.FuncAnimation(fig, update, frames=num_frames, interval=50,␣
↪blit=False)

# Suppress static plot
plt.close(fig)

# Display animation in Jupyter Notebook
HTML(ani.to_jshtml())
```

[4]: <IPython.core.display.HTML object>

[5]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
from IPython.display import HTML
import random

# Define cube vertices relative to the origin
def create_cube(size=0.5):
    half = size / 2
    vertices = np.array([
        [-half, -half, -half], [half, -half, -half], [half, half, -half],␣
↪[-half, half, -half],  # Bottom face
        [-half, -half, half], [half, -half, half], [half, half, half], [-half,␣
↪half, half]        # Top face
    ])

    faces = [
        [0, 1, 2, 3],  # Bottom face
        [4, 5, 6, 7],  # Top face
        [0, 1, 5, 4],  # Front face
        [2, 3, 7, 6],  # Back face
        [1, 2, 6, 5],  # Right face
        [4, 7, 3, 0],  # Left face
    ]

    return np.array(vertices), faces
```

```python
# Function to rotate a point around an arbitrary axis using Rodrigues' formula
def rotate_points(points, axis, theta):
    """
    Rotate points around a given axis by an angle theta (in radians).
    Uses Rodrigues' rotation formula.
    """
    axis = axis / np.linalg.norm(axis)  # Normalize axis
    cos_theta = np.cos(theta)
    sin_theta = np.sin(theta)
    cross_matrix = np.array([
        [0, -axis[2], axis[1]],
        [axis[2], 0, -axis[0]],
        [-axis[1], axis[0], 0]
    ])
    rotation_matrix = cos_theta * np.eye(3) + sin_theta * cross_matrix + (1 -␣
 ↪cos_theta) * np.outer(axis, axis)
    return np.dot(points, rotation_matrix.T)  # Apply rotation

# Generate cube
cube_vertices, cube_faces = create_cube(size=0.5)

# Define colors: Assign a random color to each face
color_choices = ['red', 'green', 'blue', 'white']
cube_colors = [random.choice(color_choices) for _ in range(6)]

# Create figure and 3D axis
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Set limits
ax.set_xlim([-1.5, 1.5])
ax.set_ylim([-1.5, 1.5])
ax.set_zlim([-1.5, 1.5])
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')

# Define start and end points
start = np.array([-1, -1, -1])
end = np.array([1, 1, 1])
num_frames = 100

# Compute evenly spaced positions for the cube's motion
positions = np.linspace(start, end, num_frames)

# Define rotation parameters
```

```
rotation_axis = end - start  # Rotate about the motion path
total_rotation = 2 * np.pi * 2  # 2 full rotations (720 degrees)
angles = np.linspace(0, total_rotation, num_frames)  # Angles for each frame

# Initialize cube poly collection
cube_poly = Poly3DCollection([], facecolors=cube_colors, edgecolor='black')

# Add cube to the axis
ax.add_collection3d(cube_poly)

# Animation update function
def update(frame):
    # Compute the new position
    position = positions[frame]

    # Rotate the cube vertices
    rotated_vertices = rotate_points(cube_vertices, rotation_axis,␣
↪angles[frame])

    # Translate cube to the new position
    transformed_vertices = rotated_vertices + position

    # Create updated cube faces
    poly_faces = [[transformed_vertices[i] for i in face] for face in␣
↪cube_faces]

    # Update the cube polygon collection
    cube_poly.set_verts(poly_faces)

    return cube_poly,

# Create animation
ani = animation.FuncAnimation(fig, update, frames=num_frames, interval=50,␣
↪blit=False)

# Suppress static plot
plt.close(fig)

# Display animation in Jupyter Notebook
HTML(ani.to_jshtml())
```

[5]: <IPython.core.display.HTML object>

[6]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
```

```python
from IPython.display import HTML
import random

# Define cube vertices relative to the origin
def create_cube(size=0.5):
    half = size / 2
    vertices = np.array([
        [-half, -half, -half], [half, -half, -half], [half, half, -half],␣
↪[-half, half, -half],  # Bottom face
        [-half, -half, half], [half, -half, half], [half, half, half], [-half,␣
↪half, half]        # Top face
    ])

    faces = [
        [0, 1, 2, 3],  # Bottom face
        [4, 5, 6, 7],  # Top face
        [0, 1, 5, 4],  # Front face
        [2, 3, 7, 6],  # Back face
        [1, 2, 6, 5],  # Right face
        [4, 7, 3, 0],  # Left face
    ]

    return np.array(vertices), faces

# Function to rotate a point around an arbitrary axis using Rodrigues' formula
def rotate_points(points, axis, theta):
    axis = axis / np.linalg.norm(axis)  # Normalize axis
    cos_theta = np.cos(theta)
    sin_theta = np.sin(theta)
    cross_matrix = np.array([
        [0, -axis[2], axis[1]],
        [axis[2], 0, -axis[0]],
        [-axis[1], axis[0], 0]
    ])
    rotation_matrix = cos_theta * np.eye(3) + sin_theta * cross_matrix + (1 -␣
↪cos_theta) * np.outer(axis, axis)
    return np.dot(points, rotation_matrix.T)  # Apply rotation

# Generate cube
cube_vertices, cube_faces = create_cube(size=0.5)

# Define colors: Assign a random color to each face
color_choices = ['red', 'green', 'blue', 'white']
cube_colors = [random.choice(color_choices) for _ in range(6)]

# Create figure and 3D axis
fig = plt.figure()
```

```python
ax = fig.add_subplot(111, projection='3d')

# Remove axis labels, grid, and ticks
ax.set_axis_off()
ax.set_xticks([])
ax.set_yticks([])
ax.set_zticks([])

# Define start and end points
start = np.array([-1, -1, -1])
end = np.array([1, 1, 1])
num_frames = 100

# Compute evenly spaced positions for the cube's motion
positions = np.linspace(start, end, num_frames)

# Define rotation parameters
rotation_axis = end - start  # Rotate about the motion path
total_rotation = 2 * np.pi * 2  # 2 full rotations (720 degrees)
angles = np.linspace(0, total_rotation, num_frames)  # Angles for each frame

# Initialize cube poly collection
cube_poly = Poly3DCollection([], facecolors=cube_colors, edgecolor='black')

# Add cube to the axis
ax.add_collection3d(cube_poly)

# Animation update function
def update(frame):
    # Compute the new position
    position = positions[frame]

    # Rotate the cube vertices
    rotated_vertices = rotate_points(cube_vertices, rotation_axis,␣
 ↪angles[frame])

    # Translate cube to the new position
    transformed_vertices = rotated_vertices + position

    # Create updated cube faces
    poly_faces = [[transformed_vertices[i] for i in face] for face in␣
 ↪cube_faces]

    # Update the cube polygon collection
    cube_poly.set_verts(poly_faces)

    return cube_poly,
```

```
# Create animation
ani = animation.FuncAnimation(fig, update, frames=num_frames, interval=50,␣
↪blit=False)

# Suppress static plot
plt.close(fig)

# Display animation in Jupyter Notebook
HTML(ani.to_jshtml())
```

[6]: <IPython.core.display.HTML object>

[7]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
from IPython.display import HTML
import random

# Define cube vertices relative to the origin
def create_cube(size=0.5):
    half = size / 2
    vertices = np.array([
        [-half, -half, -half], [half, -half, -half], [half, half, -half],␣
↪[-half, half, -half],  # Bottom face
        [-half, -half, half], [half, -half, half], [half, half, half], [-half,␣
↪half, half]        # Top face
    ])

    faces = [
        [0, 1, 2, 3],  # Bottom face
        [4, 5, 6, 7],  # Top face
        [0, 1, 5, 4],  # Front face
        [2, 3, 7, 6],  # Back face
        [1, 2, 6, 5],  # Right face
        [4, 7, 3, 0],  # Left face
    ]

    return np.array(vertices), faces

# Function to rotate a point around an arbitrary axis using Rodrigues' formula
def rotate_points(points, axis, theta):
    axis = axis / np.linalg.norm(axis)  # Normalize axis
    cos_theta = np.cos(theta)
    sin_theta = np.sin(theta)
    cross_matrix = np.array([
```

```
        [0, -axis[2], axis[1]],
        [axis[2], 0, -axis[0]],
        [-axis[1], axis[0], 0]
    ])
    rotation_matrix = cos_theta * np.eye(3) + sin_theta * cross_matrix + (1 -
 ↪cos_theta) * np.outer(axis, axis)
    return np.dot(points, rotation_matrix.T)  # Apply rotation

# Generate cube
cube_vertices, cube_faces = create_cube(size=0.5)

# Define colors: Assign a random color to each face
color_choices = ['red', 'green', 'blue', 'white']
cube_colors = [random.choice(color_choices) for _ in range(6)]

# Create figure and 3D axis
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Restore correct axis limits so animation is not clipped
ax.set_xlim([-1.5, 1.5])
ax.set_ylim([-1.5, 1.5])
ax.set_zlim([-1.5, 1.5])

# Hide axis labels, ticks, and grid while keeping limits correct
ax.set_xticks([])
ax.set_yticks([])
ax.set_zticks([])
ax.grid(False)

# Define start and end points
start = np.array([-1, -1, -1])
end = np.array([1, 1, 1])
num_frames = 100

# Compute evenly spaced positions for the cube's motion
positions = np.linspace(start, end, num_frames)

# Define rotation parameters
rotation_axis = end - start  # Rotate about the motion path
total_rotation = 2 * np.pi * 2  # 2 full rotations (720 degrees)
angles = np.linspace(0, total_rotation, num_frames)  # Angles for each frame

# Initialize cube poly collection
cube_poly = Poly3DCollection([], facecolors=cube_colors, edgecolor='black')

# Add cube to the axis
```

```
ax.add_collection3d(cube_poly)

# Animation update function
def update(frame):
    # Compute the new position
    position = positions[frame]

    # Rotate the cube vertices
    rotated_vertices = rotate_points(cube_vertices, rotation_axis,␣
 ↪angles[frame])

    # Translate cube to the new position
    transformed_vertices = rotated_vertices + position

    # Create updated cube faces
    poly_faces = [[transformed_vertices[i] for i in face] for face in␣
 ↪cube_faces]

    # Update the cube polygon collection
    cube_poly.set_verts(poly_faces)

    return cube_poly,

# Create animation
ani = animation.FuncAnimation(fig, update, frames=num_frames, interval=50,␣
 ↪blit=False)

# Suppress static plot
plt.close(fig)

# Display animation in Jupyter Notebook
HTML(ani.to_jshtml())
```

[7]: <IPython.core.display.HTML object>

[8]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
from IPython.display import HTML
import random

# Define cube vertices relative to the origin
def create_cube(size=0.5):
    half = size / 2
    vertices = np.array([
```

```
        [-half, -half, -half], [half, -half, -half], [half, half, -half],␣
↪[-half, half, -half],  # Bottom face
        [-half, -half, half], [half, -half, half], [half, half, half], [-half,␣
↪half, half]        # Top face
    ])

    faces = [
        [0, 1, 2, 3],  # Bottom face
        [4, 5, 6, 7],  # Top face
        [0, 1, 5, 4],  # Front face
        [2, 3, 7, 6],  # Back face
        [1, 2, 6, 5],  # Right face
        [4, 7, 3, 0],  # Left face
    ]

    return np.array(vertices), faces

# Function to rotate a point around an arbitrary axis using Rodrigues' formula
def rotate_points(points, axis, theta):
    axis = axis / np.linalg.norm(axis)  # Normalize axis
    cos_theta = np.cos(theta)
    sin_theta = np.sin(theta)
    cross_matrix = np.array([
        [0, -axis[2], axis[1]],
        [axis[2], 0, -axis[0]],
        [-axis[1], axis[0], 0]
    ])
    rotation_matrix = cos_theta * np.eye(3) + sin_theta * cross_matrix + (1 -␣
↪cos_theta) * np.outer(axis, axis)
    return np.dot(points, rotation_matrix.T)  # Apply rotation

# Generate cube
cube_vertices, cube_faces = create_cube(size=0.5)

# Define colors: Assign a random color to each face
color_choices = ['red', 'green', 'blue', 'white']
cube_colors = [random.choice(color_choices) for _ in range(6)]

# Create figure and 3D axis
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Restore correct axis limits so animation is not clipped
ax.set_xlim([-1.5, 1.5])
ax.set_ylim([-1.5, 1.5])
ax.set_zlim([-1.5, 1.5])
```

```python
# Hide axis labels, ticks, and grid while keeping limits correct
ax.set_xticks([])
ax.set_yticks([])
ax.set_zticks([])
ax.grid(False)

# Completely remove the axes and background cube
ax.set_proj_type('ortho')  # Orthographic projection (removes perspective distortion)
ax.axis("off")  # Hide all axis elements

# Define start and end points
start = np.array([-1, -1, -1])
end = np.array([1, 1, 1])
num_frames = 100

# Compute evenly spaced positions for the cube's motion
positions = np.linspace(start, end, num_frames)

# Define rotation parameters
rotation_axis = end - start  # Rotate about the motion path
total_rotation = 2 * np.pi * 2  # 2 full rotations (720 degrees)
angles = np.linspace(0, total_rotation, num_frames)  # Angles for each frame

# Initialize cube poly collection
cube_poly = Poly3DCollection([], facecolors=cube_colors, edgecolor='black')

# Add cube to the axis
ax.add_collection3d(cube_poly)

# Animation update function
def update(frame):
    # Compute the new position
    position = positions[frame]

    # Rotate the cube vertices
    rotated_vertices = rotate_points(cube_vertices, rotation_axis, angles[frame])

    # Translate cube to the new position
    transformed_vertices = rotated_vertices + position

    # Create updated cube faces
    poly_faces = [[transformed_vertices[i] for i in face] for face in cube_faces]

    # Update the cube polygon collection
```

```
    cube_poly.set_verts(poly_faces)

    return cube_poly,

# Create animation
ani = animation.FuncAnimation(fig, update, frames=num_frames, interval=50,␣
 ↪blit=False)

# Suppress static plot
plt.close(fig)

# Display animation in Jupyter Notebook
HTML(ani.to_jshtml())
```

[8]: <IPython.core.display.HTML object>

[9]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
from IPython.display import HTML
import random

# Define cube vertices relative to the origin
def create_cube(size=0.5):
    half = size / 2
    vertices = np.array([
        [-half, -half, -half], [half, -half, -half], [half, half, -half],␣
 ↪[-half, half, -half],  # Bottom face
        [-half, -half, half], [half, -half, half], [half, half, half], [-half,␣
 ↪half, half]        # Top face
    ])

    faces = [
        [0, 1, 2, 3],  # Bottom face
        [4, 5, 6, 7],  # Top face
        [0, 1, 5, 4],  # Front face
        [2, 3, 7, 6],  # Back face
        [1, 2, 6, 5],  # Right face
        [4, 7, 3, 0],  # Left face
    ]

    return np.array(vertices), faces

# Function to rotate a point around an arbitrary axis using Rodrigues' formula
def rotate_points(points, axis, theta):
    axis = axis / np.linalg.norm(axis)  # Normalize axis
```

```
    cos_theta = np.cos(theta)
    sin_theta = np.sin(theta)
    cross_matrix = np.array([
        [0, -axis[2], axis[1]],
        [axis[2], 0, -axis[0]],
        [-axis[1], axis[0], 0]
    ])
    rotation_matrix = cos_theta * np.eye(3) + sin_theta * cross_matrix + (1 -␣
↪cos_theta) * np.outer(axis, axis)
    return np.dot(points, rotation_matrix.T)  # Apply rotation

# Generate cube
cube_vertices, cube_faces = create_cube(size=0.5)

# Define colors: Assign a random color to each face
color_choices = ['red', 'green', 'blue', 'white']
cube_colors = [random.choice(color_choices) for _ in range(6)]

# Create figure and 3D axis
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Restore correct axis limits so animation is not clipped
ax.set_xlim([-1.5, 1.5])
ax.set_ylim([-1.5, 1.5])
ax.set_zlim([-1.5, 1.5])

# Hide axis labels, ticks, and grid while keeping limits correct
ax.set_xticks([])
ax.set_yticks([])
ax.set_zticks([])
ax.grid(False)

# Completely remove the axes and background cube
ax.set_proj_type('ortho')  # Orthographic projection (removes perspective␣
↪distortion)
ax.axis("off")  # Hide all axis elements

# Define time parameter for the great circle path
num_frames = 100
t_values = np.linspace(0, 2 * np.pi, num_frames)

# Compute the great circle path
positions = np.array([
    [np.cos(t) * np.sin(t), np.sin(t)**2, np.cos(t)]
    for t in t_values
])
```

```
# Define rotation parameters
total_rotation = 2 * np.pi * 2  # 2 full rotations (720 degrees)
angles = np.linspace(0, total_rotation, num_frames)  # Angles for each frame
rotation_axis = np.array([1, 1, 1])  # Diagonal axis through the cube center

# Initialize cube poly collection
cube_poly = Poly3DCollection([], facecolors=cube_colors, edgecolor='black')

# Add cube to the axis
ax.add_collection3d(cube_poly)

# Animation update function
def update(frame):
    # Compute the new position
    position = positions[frame]

    # Rotate the cube vertices around its own diagonal axis
    rotated_vertices = rotate_points(cube_vertices, rotation_axis,␣
 ↪angles[frame])

    # Translate cube to the new position
    transformed_vertices = rotated_vertices + position

    # Create updated cube faces
    poly_faces = [[transformed_vertices[i] for i in face] for face in␣
 ↪cube_faces]

    # Update the cube polygon collection
    cube_poly.set_verts(poly_faces)

    return cube_poly,

# Create animation
ani = animation.FuncAnimation(fig, update, frames=num_frames, interval=50,␣
 ↪blit=False)

# Suppress static plot
plt.close(fig)

# Display animation in Jupyter Notebook
HTML(ani.to_jshtml())
```

[9]: <IPython.core.display.HTML object>

[10]:
```
import numpy as np
import matplotlib.pyplot as plt
```

```python
import matplotlib.animation as animation
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
from IPython.display import HTML
import random

# Define cube vertices relative to the origin (centered at (0,0,0))
def create_cube(size=0.5):
    half = size / 2
    vertices = np.array([
        [-half, -half, -half], [half, -half, -half], [half, half, -half], [-half, half, -half],  # Bottom face
        [-half, -half, half], [half, -half, half], [half, half, half], [-half, half, half]        # Top face
    ])

    faces = [
        [0, 1, 2, 3],  # Bottom face
        [4, 5, 6, 7],  # Top face
        [0, 1, 5, 4],  # Front face
        [2, 3, 7, 6],  # Back face
        [1, 2, 6, 5],  # Right face
        [4, 7, 3, 0],  # Left face
    ]

    return np.array(vertices), faces

# Function to rotate a point around an arbitrary axis using Rodrigues' formula
def rotate_points(points, axis, theta):
    axis = axis / np.linalg.norm(axis)  # Normalize axis
    cos_theta = np.cos(theta)
    sin_theta = np.sin(theta)
    cross_matrix = np.array([
        [0, -axis[2], axis[1]],
        [axis[2], 0, -axis[0]],
        [-axis[1], axis[0], 0]
    ])
    rotation_matrix = cos_theta * np.eye(3) + sin_theta * cross_matrix + (1 - cos_theta) * np.outer(axis, axis)
    return np.dot(points, rotation_matrix.T)  # Apply rotation

# Generate cube (centered at the origin)
cube_vertices, cube_faces = create_cube(size=0.5)

# Define colors: Assign a random color to each face
color_choices = ['red', 'green', 'blue', 'white']
cube_colors = [random.choice(color_choices) for _ in range(6)]
```

```
# Create figure and 3D axis
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Restore correct axis limits so animation is not clipped
ax.set_xlim([-1.5, 1.5])
ax.set_ylim([-1.5, 1.5])
ax.set_zlim([-1.5, 1.5])

# Hide axis labels, ticks, and grid while keeping limits correct
ax.set_xticks([])
ax.set_yticks([])
ax.set_zticks([])
ax.grid(False)

# Completely remove the axes and background cube
ax.set_proj_type('ortho')  # Orthographic projection (removes perspective␣
↪distortion)
ax.axis("off")  # Hide all axis elements

# Define time parameter for the great circle path
num_frames = 100
t_values = np.linspace(0, 2 * np.pi, num_frames)

# Compute the great circle path for the cube center
positions = np.array([
    [np.cos(t) * np.sin(t), np.sin(t)**2, np.cos(t)]
    for t in t_values
])

# Define rotation parameters
total_rotation = 2 * np.pi * 2  # 2 full rotations (720 degrees)
angles = np.linspace(0, total_rotation, num_frames)  # Angles for each frame

# The rotation axis should pass through the cube center
rotation_axis = np.array([1, 1, 1]) / np.sqrt(3)  # Normalized diagonal axis

# Initialize cube poly collection
cube_poly = Poly3DCollection([], facecolors=cube_colors, edgecolor='black')

# Add cube to the axis
ax.add_collection3d(cube_poly)

# Animation update function
def update(frame):
    # Compute the new position (center of the cube)
    cube_center = positions[frame]
```

```
    # Rotate the cube vertices around its own diagonal axis
    rotated_vertices = rotate_points(cube_vertices, rotation_axis,␣
↪angles[frame])

    # Translate cube to follow the great circle path **from its center**
    transformed_vertices = rotated_vertices + cube_center

    # Create updated cube faces
    poly_faces = [[transformed_vertices[i] for i in face] for face in␣
↪cube_faces]

    # Update the cube polygon collection
    cube_poly.set_verts(poly_faces)

    return cube_poly,

# Create animation
ani = animation.FuncAnimation(fig, update, frames=num_frames, interval=50,␣
↪blit=False)

# Suppress static plot
plt.close(fig)

# Display animation in Jupyter Notebook
HTML(ani.to_jshtml())
```

[10]: <IPython.core.display.HTML object>

[11]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
from IPython.display import HTML
import random

# Define cube vertices relative to the origin (centered at (0,0,0))
def create_cube(size=0.5):
    half = size / 2
    vertices = np.array([
        [-half, -half, -half], [half, -half, -half], [half, half, -half],␣
↪[-half, half, -half],  # Bottom face
        [-half, -half, half], [half, -half, half], [half, half, half], [-half,␣
↪half, half]        # Top face
    ])

    faces = [
```

```
        [0, 1, 2, 3],  # Bottom face
        [4, 5, 6, 7],  # Top face
        [0, 1, 5, 4],  # Front face
        [2, 3, 7, 6],  # Back face
        [1, 2, 6, 5],  # Right face
        [4, 7, 3, 0],  # Left face
    ]

    return np.array(vertices), faces

# Function to rotate a point around an arbitrary axis using Rodrigues' formula
def rotate_points(points, axis, theta):
    axis = axis / np.linalg.norm(axis)  # Normalize axis
    cos_theta = np.cos(theta)
    sin_theta = np.sin(theta)
    cross_matrix = np.array([
        [0, -axis[2], axis[1]],
        [axis[2], 0, -axis[0]],
        [-axis[1], axis[0], 0]
    ])
    rotation_matrix = cos_theta * np.eye(3) + sin_theta * cross_matrix + (1 -
 ↪cos_theta) * np.outer(axis, axis)
    return np.dot(points, rotation_matrix.T)  # Apply rotation

# Generate cube (centered at the origin)
cube_vertices, cube_faces = create_cube(size=0.5)

# Define colors: Assign a random color to each face, using lighter blue
color_choices = ['red', 'green', 'lightskyblue', 'white']
cube_colors = [random.choice(color_choices) for _ in range(6)]

# Create figure and 3D axis
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Restore correct axis limits so animation is not clipped
ax.set_xlim([-1.5, 1.5])
ax.set_ylim([-1.5, 1.5])
ax.set_zlim([-1.5, 1.5])

# Hide axis labels, ticks, and grid while keeping limits correct
ax.set_xticks([])
ax.set_yticks([])
ax.set_zticks([])
ax.grid(False)

# Completely remove the axes and background cube
```

```
ax.set_proj_type('ortho')  # Orthographic projection (removes perspective␣
↪distortion)
ax.axis("off")  # Hide all axis elements

# Define time parameter for the great circle path
num_frames = 100
t_values = np.linspace(0, 2 * np.pi, num_frames)

# Compute the great circle path
raw_positions = np.array([
    [np.cos(t) * np.sin(t), np.sin(t)**2, np.cos(t)]
    for t in t_values
])

# Center the path around (0,0,0)
positions = raw_positions - np.mean(raw_positions, axis=0)

# Define rotation parameters
total_rotation = 2 * np.pi * 2  # 2 full rotations (720 degrees)
angles = np.linspace(0, total_rotation, num_frames)  # Angles for each frame

# The rotation axis should pass through the cube center
rotation_axis = np.array([1, 1, 1]) / np.sqrt(3)  # Normalized diagonal axis

# Initialize cube poly collection
cube_poly = Poly3DCollection([], facecolors=cube_colors, edgecolor='black')

# Add cube to the axis
ax.add_collection3d(cube_poly)

# Animation update function
def update(frame):
    # Compute the new position (center of the cube)
    cube_center = positions[frame]

    # Rotate the cube vertices around its own diagonal axis
    rotated_vertices = rotate_points(cube_vertices, rotation_axis,␣
↪angles[frame])

    # Translate cube to follow the great circle path **from its center**
    transformed_vertices = rotated_vertices + cube_center

    # Create updated cube faces
    poly_faces = [[transformed_vertices[i] for i in face] for face in␣
↪cube_faces]

    # Update the cube polygon collection
```

```python
    cube_poly.set_verts(poly_faces)

    return cube_poly,

# Create animation
ani = animation.FuncAnimation(fig, update, frames=num_frames, interval=50,␣
 ↪blit=False)

# Suppress static plot
plt.close(fig)

# Display animation in Jupyter Notebook
HTML(ani.to_jshtml())
```

[11]: <IPython.core.display.HTML object>

[12]:
```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
from IPython.display import HTML
import random

# Define cube vertices relative to the origin (centered at (0,0,0))
def create_cube(size=0.5):
    half = size / 2
    vertices = np.array([
        [-half, -half, -half], [half, -half, -half], [half, half, -half],␣
 ↪[-half, half, -half],  # Bottom face
        [-half, -half, half], [half, -half, half], [half, half, half], [-half,␣
 ↪half, half]        # Top face
    ])

    faces = [
        [0, 1, 2, 3],  # Bottom face
        [4, 5, 6, 7],  # Top face
        [0, 1, 5, 4],  # Front face
        [2, 3, 7, 6],  # Back face
        [1, 2, 6, 5],  # Right face
        [4, 7, 3, 0],  # Left face
    ]

    return np.array(vertices), faces

# Function to rotate a point around an arbitrary axis using Rodrigues' formula
def rotate_points(points, axis, theta):
    axis = axis / np.linalg.norm(axis)  # Normalize axis
```

```python
    cos_theta = np.cos(theta)
    sin_theta = np.sin(theta)
    cross_matrix = np.array([
        [0, -axis[2], axis[1]],
        [axis[2], 0, -axis[0]],
        [-axis[1], axis[0], 0]
    ])
    rotation_matrix = cos_theta * np.eye(3) + sin_theta * cross_matrix + (1 -␣
↪cos_theta) * np.outer(axis, axis)
    return np.dot(points, rotation_matrix.T)  # Apply rotation

# Generate cube (centered at the origin)
cube_vertices, cube_faces = create_cube(size=1.0)

# Define colors: Assign a random color to each face, using a lighter blue
color_choices = ['red', 'forestgreen', 'dodgerblue', 'white']
cube_colors = [random.choice(color_choices) for _ in range(6)]

# Create figure and 3D axis
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Restore correct axis limits so animation is not clipped
ax.set_xlim([-1.5, 1.5])
ax.set_ylim([-1.5, 1.5])
ax.set_zlim([-1.5, 1.5])

# Hide axis labels, ticks, and grid while keeping limits correct
ax.set_xticks([])
ax.set_yticks([])
ax.set_zticks([])
ax.grid(False)

# Completely remove the axes and background cube
#ax.set_proj_type('ortho')  # Orthographic projection (removes perspective␣
↪distortion)
ax.axis("off")  # Hide all axis elements

# Define time parameter for the great circle path
num_frames = 100
t_values = np.linspace(0, 2 * np.pi, num_frames)

# Compute the great circle path
a = np.array([0.70710678, -0.70710678, 0.0])  # First vector in plane
b = np.array([0.40824829, 0.40824829, 0.81649658])  # Second vector in plane

positions = np.array([
```

```
    a * np.cos(t) + b * np.sin(t) for t in t_values
])

# Define rotation parameters
total_rotation = 2 * np.pi * 2  # 2 full rotations (720 degrees)
angles = np.linspace(0, total_rotation, num_frames)  # Angles for each frame

# The rotation axis should pass through the cube center
rotation_axis = np.array([1, 1, 1]) / np.sqrt(3)  # Normalized diagonal axis

# Initialize cube poly collection
cube_poly = Poly3DCollection([], facecolors=cube_colors, edgecolor='black')

# Add cube to the axis
ax.add_collection3d(cube_poly)

# Animation update function
def update(frame):
    # Compute the new position (center of the cube)
    cube_center = positions[frame]

    # Rotate the cube vertices around its own diagonal axis
    rotated_vertices = rotate_points(cube_vertices, rotation_axis,␣
 ↪angles[frame])

    # Translate cube to follow the great circle path **from its center**
    transformed_vertices = rotated_vertices + cube_center

    # Create updated cube faces
    poly_faces = [[transformed_vertices[i] for i in face] for face in␣
 ↪cube_faces]

    # Update the cube polygon collection
    cube_poly.set_verts(poly_faces)

    return cube_poly,

# Create animation
ani = animation.FuncAnimation(fig, update, frames=num_frames, interval=50,␣
 ↪blit=False)

# Suppress static plot
plt.close(fig)

# Display animation in Jupyter Notebook
HTML(ani.to_jshtml())
```

```
[12]: <IPython.core.display.HTML object>
```