Orthogonal Vectors Generator and Visualizer

Documentation

March 4, 2025

Abstract

This document provides a mathematical explanation and implementation details for generating and visualizing three orthogonal vectors from a given origin point. The document covers the mathematical formulation, implementation in Python, and visualization techniques.

Contents

1 Introduction

In three-dimensional space, orthogonal vectors are perpendicular to each other, meaning their dot product equals zero. This document describes a method to generate three orthogonal vectors from a given origin point $\vec{R}_0$ and visualize them in both 3D and 2D projections.

2 Mathematical Formulation

2.1 Definitions

Let $\vec{R}_0$ be the origin vector, typically set to $(0, 0, 0)$. We define three orthogonal vectors $\vec{R}_1$, $\vec{R}_2$, and $\vec{R}_3$ as follows:

$$\vec{R}_1 = \vec{R}_0 + d \cdot \cos(\theta) \cdot \sqrt{\frac{2}{3}} \cdot (1, -\frac{1}{2}, -\frac{1}{2}) \tag{1}$$

$$\vec{R}_2 = \vec{R}_0 + d \cdot \frac{\cos(\theta)/\sqrt{3} + \sin(\theta)}{\sqrt{2}} \cdot (1, 1, 1) \tag{2}$$

$$\vec{R}_3 = \vec{R}_0 + d \cdot \frac{\sin(\theta) - \cos(\theta)/\sqrt{3}}{\sqrt{2}} \cdot \sqrt{2} \cdot (0, -\frac{1}{2}, \frac{1}{2}) \tag{3}$$

where:

- d is a distance parameter that scales the vectors
- θ is an angle parameter that rotates the vectors

2.2 Verification of Orthogonality

For vectors to be orthogonal, their dot product must be zero. Let's verify this property for our vectors:

$$(\vec{R}_1 - \vec{R}_0) \cdot (\vec{R}_2 - \vec{R}_0) = 0 \tag{4}$$

$$(\vec{R}_1 - \vec{R}_0) \cdot (\vec{R}_3 - \vec{R}_0) = 0 \tag{5}$$

$$(\vec{R}_2 - \vec{R}_0) \cdot (\vec{R}_3 - \vec{R}_0) = 0 \tag{6}$$

The orthogonality is maintained regardless of the values of d and θ.

3 Implementation

3.1 Python Code

The implementation uses Python with NumPy for vector operations and Matplotlib for visualization. Here's the core function that generates the orthogonal vectors:

```python
def create_orthogonal_vectors(R_0=(0, 0, 0), d=1, theta=0):
    """
    Create 3 orthogonal R vectors for R_0

    Parameters:
    R_0 (tuple): The origin vector, default is (0, 0, 0)
    d (float): The distance parameter, default is 1
    theta (float): The angle parameter in radians, default is 0

    Returns:
    tuple: Three orthogonal vectors R_1, R_2, R_3
    """
    # Convert R_0 to numpy array for vector operations
    R_0 = np.array(R_0)

    # Calculate R_1, R_2, R_3 according to the given formulas
    # R_1 = R_0 + d * (cos(theta))*sqrt(2/3)
    R_1 = R_0 + d * np.cos(theta) * np.sqrt(2/3) * np.array([1,
    -1/2, -1/2])

    # R_2 = R_0 + d * (cos(theta)/sqrt(3) + sin(theta))/sqrt(2)
    R_2 = R_0 + d * (np.cos(theta)/np.sqrt(3) + np.sin(theta))/np.
    sqrt(2) * np.array([1, 1, 1])

    # R_3 = R_0 + d * (sin(theta) - cos(theta)/sqrt(3))/sqrt(2)
    R_3 = R_0 + d * (np.sin(theta) - np.cos(theta)/np.sqrt(3))/np.
    sqrt(2) * np.array([0, -1/2, 1/2]) * np.sqrt(2)

    return R_1, R_2, R_3
```

Listing 1: Orthogonal Vectors Generation

3.2 Visualization Techniques

The implementation includes several visualization techniques:

3.2.1 3D Visualization

The 3D visualization uses Matplotlib's 3D plotting capabilities to display all three vectors from the origin in three-dimensional space.

3.2.2 2D Projections

Four different 2D projections are provided:

- XY Plane: Projection onto the plane where z=0
- XZ Plane: Projection onto the plane where y=0
- YZ Plane: Projection onto the plane where x=0
- $\vec{R}_0$ Plane: Projection onto a plane passing through $\vec{R}_0$ and perpendicular to the vector from the origin to $\vec{R}_0$

4 Usage Examples

4.1 Basic Usage

To generate and visualize the orthogonal vectors with default parameters:

```
# Define parameters
R_0 = np.array([0, 0, 0])  # Origin
d = 1                      # Distance parameter
theta = math.pi/4          # 45 degrees in radians

# Create the orthogonal vectors
R_1, R_2, R_3 = create_orthogonal_vectors(R_0, d, theta)

# Check orthogonality
dot_1_2 = np.dot(R_1 - R_0, R_2 - R_0)
dot_1_3 = np.dot(R_1 - R_0, R_3 - R_0)
dot_2_3 = np.dot(R_2 - R_0, R_3 - R_0)

print("R_1 \cdot R_2:", dot_1_2)  # Should be close to 0
print("R_1 \cdot R_3:", dot_1_3)  # Should be close to 0
print("R_2 \cdot R_3:", dot_2_3)  # Should be close to 0
```

Listing 2: Basic Usage Example

4.2 Customization

The vectors can be customized by modifying the parameters:

- $\vec{R}_0$: Change the origin point
- d: Adjust the scale of the vectors
- θ: Rotate the vectors around the origin

5 Mathematical Properties

5.1 Invariance to Rotation

The orthogonality of the vectors is preserved regardless of the value of θ. This means that the vectors can be rotated around the origin while maintaining their perpendicular relationship.

5.2 Scaling

The parameter d scales all vectors equally, preserving their orthogonality. This allows for adjusting the size of the vector system without changing its geometric properties.

6 Conclusion

This document has presented a method for generating and visualizing three orthogonal vectors from a given origin point. The mathematical formulation ensures that the vectors remain orthogonal regardless of the parameter values, and the Python implementation provides both 3D and 2D visualizations to help understand the geometric relationships between the vectors.

A Complete Code Listing

The complete Python implementation can be found in the main.py file in the project root directory.