

Generalized Orthogonal Vectors Generator and Visualizer

Technical Documentation

March 4, 2025

Abstract

This document provides a comprehensive guide to the generalized implementation of the Orthogonal Vectors Generator and Visualizer. The implementation offers a modular and configurable approach to creating and visualizing three orthogonal vectors from a given origin point. This document covers the mathematical formulation, implementation details, API reference, usage examples, and visualization techniques.

Contents

1	Introduction	4
1.1	Project Overview	4
1.2	Key Features	4
1.3	Package Structure	4
1.4	Document Structure	4
2	Mathematical Formulation	5
2.1	Definitions	5
2.2	Verification of Orthogonality	5
2.3	Mathematical Properties	6
2.3.1	Invariance to Origin	6
2.3.2	Invariance to Rotation	6
2.3.3	Scaling	6
2.4	Geometric Interpretation	6
3	Implementation	6
3.1	Modular Architecture	6
3.2	Vector Utilities	7
3.3	Visualization	7
3.4	Configuration Management	7
3.5	Command-line Interface	7
3.6	Package Initialization	8
3.7	Example Scripts	8
3.8	Dependencies	8
4	API Reference	8
4.1	vector_utils Module	8
4.1.1	create_orthogonal_vectors	8
4.1.2	check_orthogonality	8
4.1.3	calculate_displacement_vectors	9
4.1.4	calculate_dot_products	9
4.2	visualization Module	9
4.2.1	plot_vectors_3d	9
4.2.2	plot_vectors_2d	10
4.2.3	plot_vectors	10
4.3	config Module	10

4.3.1	VectorConfig Class	10
4.3.2	VectorConfig.save_to_file	11
4.3.3	VectorConfig.load_from_file	11
4.4	main Module	11
4.4.1	main Function	11
4.5	__init__ Module	11
5	Usage Examples	12
5.1	Basic Usage as a Python Module	12
5.2	Customizing Vector Generation	12
5.3	Using the VectorConfig Class	12
5.4	Saving and Loading Configurations	13
5.5	Checking Orthogonality	13
5.6	Using the Command-line Interface	14
5.6.1	Basic Usage	14
5.6.2	Customizing Vector Generation	14
5.6.3	Customizing Visualization	14
5.6.4	Using a Configuration File	14
5.6.5	Saving a Configuration File	14
5.7	Complete Example Script	14
6	Visualization	15
6.1	3D Visualization	15
6.2	2D Projections	16
6.3	Customization Options	17
6.4	R0 Plane Projections	18
6.5	Implementation Details	19
6.6	Visualization in the Command-line Interface	19
7	Configuration	19
7.1	VectorConfig Class	19
7.2	Initializing a Configuration	20
7.3	Using a Configuration	20
7.4	Saving a Configuration to a File	20
7.5	Loading a Configuration from a File	20
7.6	Configuration in the Command-line Interface	21
7.7	Configuration File Format	21
7.8	Default Configuration	21
8	Command-line Interface	21
8.1	Basic Usage	22
8.2	Command-line Options	22
8.3	Examples	22
8.3.1	Customizing Vector Generation	22
8.3.2	Customizing Visualization	22
8.3.3	Using a Configuration File	23
8.3.4	Saving a Configuration File	23
8.3.5	Checking Orthogonality	23
8.3.6	Verbose Output	23
8.4	Implementation Details	23
8.5	Error Handling	23
8.6	Help Message	24
9	Example Results	24
9.1	Default Configuration	24
9.1.1	Vector Coordinates	24
9.1.2	Dot Products	24
9.1.3	3D Visualization	25
9.1.4	2D Projections	26

9.2	Custom Configuration 1	26
9.2.1	Vector Coordinates	26
9.2.2	Dot Products	27
9.2.3	3D Visualization	27
9.2.4	2D Projections	28
9.3	Custom Configuration 2	28
9.3.1	Vector Coordinates	28
9.3.2	Dot Products	29
9.3.3	3D Visualization	29
9.3.4	2D Projections	30
9.4	Effect of Distance Parameter	30
9.5	Effect of Angle Parameter	33
9.6	Effect of Origin	36
9.6.1	3D Visualizations	37
9.6.2	R_0 Plane Projections	39
9.6.3	Combined 3D and R_0 Plane Views	42
9.7	Summary of Results	44
10	Conclusion	44
10.1	Summary of Features	44
10.2	Potential Applications	45
10.3	Future Work	45
10.4	Conclusion	45
A	Source Code	46
A.1	main.py	46
A.2	vector_utils.py	48
A.3	config.py	49
A.4	visualization.py	50
A.5	__init__.py	51

1 Introduction

In three-dimensional space, orthogonal vectors are perpendicular to each other, meaning their dot product equals zero. This document describes a generalized implementation for generating and visualizing three orthogonal vectors from a given origin point.

1.1 Project Overview

The Generalized Orthogonal Vectors Generator and Visualizer is a Python package that provides a modular and configurable approach to vector generation and visualization. It is designed to be flexible, extensible, and easy to use, both as a command-line tool and as a Python package.

1.2 Key Features

The generalized implementation offers the following key features:

- **Modular Architecture:** The implementation is divided into separate modules for vector calculations, visualization, and configuration management.
- **Configurability:** All aspects of vector generation and visualization can be configured through a unified configuration system.
- **Command-line Interface:** A comprehensive command-line interface allows for easy use without writing Python code.
- **Configuration File Support:** Configurations can be saved to and loaded from JSON files.
- **Multiple Visualization Options:** Both 3D visualization and various 2D projections are supported.
- **Plot Saving:** Plots can be saved to files instead of being displayed interactively.
- **Python Package:** The implementation can be used as a Python package, allowing for integration into other projects.

1.3 Package Structure

The package is organized into the following modules:

- `vector_utils.py`: Contains functions for vector calculations, including the creation of orthogonal vectors and checking their orthogonality.
- `visualization.py`: Provides functions for visualizing vectors in 3D and 2D projections.
- `config.py`: Implements a configuration management system with the `VectorConfig` class.
- `main.py`: Provides a command-line interface for the package.
- `example.py`: Contains example scripts demonstrating different use cases.
- `__init__.py`: Package initialization and exports.

1.4 Document Structure

This document is organized as follows:

- **Mathematical Formulation:** Explains the mathematical basis for generating orthogonal vectors.
- **Implementation:** Describes the implementation details of the package.
- **API Reference:** Provides a reference for the package's API.
- **Usage Examples:** Shows examples of how to use the package.
- **Visualization:** Explains the visualization techniques used.

- **Configuration:** Describes the configuration management system.
- **Command-line Interface:** Documents the command-line interface.
- **Example Results:** Shows example results for different configurations.
- **Conclusion:** Summarizes the document and discusses potential future work.
- **Appendix:** Contains additional information, including complete code listings.

2 Mathematical Formulation

This section describes the mathematical basis for generating three orthogonal vectors from a given origin point.

2.1 Definitions

Let \vec{R}_0 be the origin vector in three-dimensional space. We define three orthogonal vectors \vec{R}_1 , \vec{R}_2 , and \vec{R}_3 as follows:

$$\vec{R}_1 = \vec{R}_0 + d \cdot \cos(\theta) \cdot \sqrt{\frac{2}{3}} \cdot \begin{pmatrix} 1 \\ -\frac{1}{2} \\ -\frac{1}{2} \end{pmatrix} \quad (1)$$

$$\vec{R}_2 = \vec{R}_0 + d \cdot \frac{\cos(\theta)/\sqrt{3} + \sin(\theta)}{\sqrt{2}} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \quad (2)$$

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

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 define the displacement vectors:

$$\vec{v}_1 = \vec{R}_1 - \vec{R}_0 = d \cdot \cos(\theta) \cdot \sqrt{\frac{2}{3}} \cdot \begin{pmatrix} 1 \\ -\frac{1}{2} \\ -\frac{1}{2} \end{pmatrix} \quad (4)$$

$$\vec{v}_2 = \vec{R}_2 - \vec{R}_0 = d \cdot \frac{\cos(\theta)/\sqrt{3} + \sin(\theta)}{\sqrt{2}} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \quad (5)$$

$$\vec{v}_3 = \vec{R}_3 - \vec{R}_0 = d \cdot \frac{\sin(\theta) - \cos(\theta)/\sqrt{3}}{\sqrt{2}} \cdot \sqrt{2} \cdot \begin{pmatrix} 0 \\ -\frac{1}{2} \\ \frac{1}{2} \end{pmatrix} \quad (6)$$

Now, let's verify that these vectors are orthogonal by calculating their dot products:

$$\vec{v}_1 \cdot \vec{v}_2 = d \cdot \cos(\theta) \cdot \sqrt{\frac{2}{3}} \cdot d \cdot \frac{\cos(\theta)/\sqrt{3} + \sin(\theta)}{\sqrt{2}} \cdot \left(1 + \left(-\frac{1}{2} \right) + \left(-\frac{1}{2} \right) \right) \quad (7)$$

$$= d^2 \cdot \cos(\theta) \cdot \sqrt{\frac{2}{3}} \cdot \frac{\cos(\theta)/\sqrt{3} + \sin(\theta)}{\sqrt{2}} \cdot 0 \quad (8)$$

$$= 0 \quad (9)$$

$$\vec{v}_1 \cdot \vec{v}_3 = d \cdot \cos(\theta) \cdot \sqrt{\frac{2}{3}} \cdot d \cdot \frac{\sin(\theta) - \cos(\theta)/\sqrt{3}}{\sqrt{2}} \cdot \sqrt{2} \cdot \left(0 + \left(-\frac{1}{2}\right) \cdot \left(-\frac{1}{2}\right) + \left(-\frac{1}{2}\right) \cdot \frac{1}{2}\right) \quad (10)$$

$$= d^2 \cdot \cos(\theta) \cdot \sqrt{\frac{2}{3}} \cdot \frac{\sin(\theta) - \cos(\theta)/\sqrt{3}}{\sqrt{2}} \cdot \sqrt{2} \cdot \left(0 + \frac{1}{4} - \frac{1}{4}\right) \quad (11)$$

$$= 0 \quad (12)$$

$$\vec{v}_2 \cdot \vec{v}_3 = d \cdot \frac{\cos(\theta)/\sqrt{3} + \sin(\theta)}{\sqrt{2}} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \cdot d \cdot \frac{\sin(\theta) - \cos(\theta)/\sqrt{3}}{\sqrt{2}} \cdot \sqrt{2} \cdot \begin{pmatrix} 0 \\ -\frac{1}{2} \\ \frac{1}{2} \end{pmatrix} \quad (13)$$

$$= d^2 \cdot \frac{\cos(\theta)/\sqrt{3} + \sin(\theta)}{\sqrt{2}} \cdot \frac{\sin(\theta) - \cos(\theta)/\sqrt{3}}{\sqrt{2}} \cdot \sqrt{2} \cdot \left(0 + 1 \cdot \left(-\frac{1}{2}\right) + 1 \cdot \frac{1}{2}\right) \quad (14)$$

$$= d^2 \cdot \frac{\cos(\theta)/\sqrt{3} + \sin(\theta)}{\sqrt{2}} \cdot \frac{\sin(\theta) - \cos(\theta)/\sqrt{3}}{\sqrt{2}} \cdot \sqrt{2} \cdot 0 \quad (15)$$

$$= 0 \quad (16)$$

The dot products are all zero, confirming that the vectors are orthogonal. This orthogonality is maintained regardless of the values of d , θ , or \vec{R}_0 .

2.3 Mathematical Properties

2.3.1 Invariance to Origin

The orthogonality of the vectors is preserved regardless of the origin point \vec{R}_0 . This is because the orthogonality depends only on the displacement vectors \vec{v}_1 , \vec{v}_2 , and \vec{v}_3 , which are independent of \vec{R}_0 .

2.3.2 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.

2.3.3 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.

2.4 Geometric Interpretation

Geometrically, the vectors \vec{R}_1 , \vec{R}_2 , and \vec{R}_3 form a right-handed orthogonal coordinate system centered at \vec{R}_0 . The parameter d controls the scale of this coordinate system, while θ controls its orientation.

The displacement vectors \vec{v}_1 , \vec{v}_2 , and \vec{v}_3 form a basis for three-dimensional space, meaning that any vector in three-dimensional space can be expressed as a linear combination of these vectors.

3 Implementation

The generalized implementation of the Orthogonal Vectors Generator and Visualizer is designed to be modular, configurable, and easy to use. This section describes the implementation details of the package.

3.1 Modular Architecture

The package is organized into the following modules:

- `vector_utils.py`: Contains functions for vector calculations.
- `visualization.py`: Provides functions for visualizing vectors.

- `config.py`: Implements a configuration management system.
- `main.py`: Provides a command-line interface.
- `example.py`: Contains example scripts.
- `__init__.py`: Package initialization and exports.

This modular architecture allows for easy maintenance, extension, and reuse of the code.

3.2 Vector Utilities

The `vector_utils.py` module provides functions for vector calculations, including:

- `create_orthogonal_vectors`: Creates three orthogonal vectors from a given origin point.
- `check_orthogonality`: Checks if a set of vectors is orthogonal.
- `calculate_displacement_vectors`: Calculates the displacement vectors from the origin.
- `calculate_dot_products`: Calculates the dot products between vectors.

The `create_orthogonal_vectors` function implements the mathematical formulation described in the previous section. It takes the origin point, distance parameter, and angle parameter as inputs and returns the three orthogonal vectors.

3.3 Visualization

The `visualization.py` module provides functions for visualizing vectors, including:

- `plot_vectors_3d`: Plots vectors in 3D space.
- `plot_vectors_2d`: Plots vectors in various 2D projections.
- `plot_vectors`: A high-level function that plots vectors in either 3D or 2D, depending on the configuration.

The visualization functions use Matplotlib to create the plots. The 3D visualization shows the vectors in three-dimensional space, while the 2D visualizations show projections onto the XY, XZ, and YZ planes, as well as a projection onto the plane containing the origin point.

3.4 Configuration Management

The `config.py` module implements a configuration management system with the `VectorConfig` class. This class provides a unified way to configure all aspects of vector generation and visualization, including:

- Origin point
- Distance parameter
- Angle parameter
- Visualization type (3D or 2D)
- Plot title and labels
- Plot saving options

The `VectorConfig` class also provides methods for saving configurations to and loading configurations from JSON files, making it easy to reuse configurations across different runs.

3.5 Command-line Interface

The `main.py` module provides a command-line interface for the package. This interface allows users to generate and visualize orthogonal vectors without writing Python code. The command-line interface supports all the configuration options provided by the `VectorConfig` class, as well as additional options for controlling the behavior of the program.

3.6 Package Initialization

The `__init__.py` module initializes the package and exports the key functions and classes, making them available when the package is imported. This makes it easy to use the package in other Python projects.

3.7 Example Scripts

The `example.py` module contains example scripts demonstrating different use cases of the package. These examples serve as a starting point for users who want to use the package in their own projects.

3.8 Dependencies

The package depends on the following Python libraries:

- `numpy`: For numerical computations
- `matplotlib`: For visualization
- `json`: For configuration file handling
- `argparse`: For command-line interface

These dependencies are specified in the `requirements.txt` file, making it easy to install them using pip.

4 API Reference

This section provides a reference for the API of the Generalized Orthogonal Vectors Generator and Visualizer package. It describes the functions and classes provided by each module.

4.1 vector_utils Module

4.1.1 create_orthogonal_vectors

```

1 def create_orthogonal_vectors(origin, d=1.0, theta=math.pi/4):
2     """
3         Create three orthogonal vectors from a given origin point.
4
5     Args:
6         origin (list or numpy.ndarray): The origin point as a 3D vector [x, y, z]
7         d (float, optional): Distance parameter. Defaults to 1.0.
8         theta (float, optional): Angle parameter in radians. Defaults to pi/4.
9
10    Returns:
11        tuple: Three orthogonal vectors (R1, R2, R3) as numpy arrays
12    """

```

This function creates three orthogonal vectors from a given origin point using the mathematical formulation described in the Mathematical Formulation section. It takes the origin point, distance parameter, and angle parameter as inputs and returns the three orthogonal vectors.

4.1.2 check_orthogonality

```

1 def check_orthogonality(vectors, origin=None, tolerance=1e-10):
2     """
3         Check if a set of vectors is orthogonal.
4
5     Args:
6         vectors (list): List of vectors to check
7         origin (list or numpy.ndarray, optional): Origin point. If provided,
8                                         checks orthogonality of
9                                         displacement vectors.
10        tolerance (float, optional): Tolerance for floating-point comparison.
11                                         Defaults to 1e-10.
12

```

```

13     Returns:
14         bool: True if vectors are orthogonal, False otherwise
15     """

```

This function checks if a set of vectors is orthogonal by calculating the dot products between them. If an origin point is provided, it checks the orthogonality of the displacement vectors from the origin. It returns True if the vectors are orthogonal (within the specified tolerance) and False otherwise.

4.1.3 calculate_displacement_vectors

```

1 def calculate_displacement_vectors(vectors, origin):
2     """
3         Calculate the displacement vectors from the origin.
4
5     Args:
6         vectors (list): List of vectors
7         origin (list or numpy.ndarray): Origin point
8
9     Returns:
10        list: List of displacement vectors
11    """

```

This function calculates the displacement vectors from the origin to each of the given vectors. It takes a list of vectors and an origin point as inputs and returns a list of displacement vectors.

4.1.4 calculate_dot_products

```

1 def calculate_dot_products(vectors):
2     """
3         Calculate the dot products between all pairs of vectors.
4
5     Args:
6         vectors (list): List of vectors
7
8     Returns:
9         list: List of dot products
10    """

```

This function calculates the dot products between all pairs of vectors in the given list. It takes a list of vectors as input and returns a list of dot products.

4.2 visualization Module

4.2.1 plot_vectors_3d

```

1 def plot_vectors_3d(vectors, origin=None, title="Orthogonal Vectors (3D)",
2                      show_plot=True, save_path=None):
3     """
4         Plot vectors in 3D space.
5
6     Args:
7         vectors (list): List of vectors to plot
8         origin (list or numpy.ndarray, optional): Origin point.
9             Defaults to [0, 0, 0].
10        title (str, optional): Plot title. Defaults to "Orthogonal Vectors (3D)".
11        show_plot (bool, optional): Whether to show the plot. Defaults to True.
12        save_path (str, optional): Path to save the plot. Defaults to None.
13
14    Returns:
15        matplotlib.figure.Figure: The figure object
16    """

```

This function plots vectors in 3D space using Matplotlib. It takes a list of vectors, an optional origin point, a title, and options for showing and saving the plot as inputs. It returns the Matplotlib figure object.

4.2.2 plot_vectors_2d

```

1 def plot_vectors_2d(vectors, origin=None,
2                     title="Orthogonal Vectors (2D Projections)",
3                     show_plot=True, save_path=None):
4     """
5         Plot vectors in various 2D projections.
6
7     Args:
8         vectors (list): List of vectors to plot
9         origin (list or numpy.ndarray, optional): Origin point.
10            Defaults to [0, 0, 0].
11
12        title (str, optional): Plot title.
13            Defaults to "Orthogonal Vectors (2D Projections)".
14
15        show_plot (bool, optional): Whether to show the plot. Defaults to True.
16
17        save_path (str, optional): Path to save the plot. Defaults to None.
18
19    Returns:
20        matplotlib.figure.Figure: The figure object
21    """

```

This function plots vectors in various 2D projections using Matplotlib. It creates four subplots showing projections onto the XY, XZ, and YZ planes, as well as a projection onto the plane containing the origin point. It takes a list of vectors, an optional origin point, a title, and options for showing and saving the plot as inputs. It returns the Matplotlib figure object.

4.2.3 plot_vectors

```

1 def plot_vectors(vectors, origin=None, plot_type="3d", title=None,
2                  show_plot=True, save_path=None):
3     """
4         Plot vectors in either 3D or 2D, depending on the plot_type.
5
6     Args:
7         vectors (list): List of vectors to plot
8         origin (list or numpy.ndarray, optional): Origin point.
9            Defaults to [0, 0, 0].
10
11        plot_type (str, optional): Type of plot, either "3d" or "2d".
12            Defaults to "3d".
13
14        title (str, optional): Plot title. Defaults to None.
15
16        show_plot (bool, optional): Whether to show the plot. Defaults to True.
17
18        save_path (str, optional): Path to save the plot. Defaults to None.
19
20    Returns:
21        matplotlib.figure.Figure: The figure object
22    """

```

This function is a high-level function that plots vectors in either 3D or 2D, depending on the specified plot type. It calls either `plot_vectors_3d` or `plot_vectors_2d` based on the `plot_type` parameter. It takes a list of vectors, an optional origin point, a plot type, a title, and options for showing and saving the plot as inputs. It returns the Matplotlib figure object.

4.3 config Module

4.3.1 VectorConfig Class

```

1 class VectorConfig:
2     """
3         Configuration class for vector generation and visualization.
4     """
5
6     def __init__(self, origin=None, d=1.0, theta=math.pi/4, plot_type="3d",
7                  title=None, show_plot=True, save_path=None):
8         """
9             Initialize the configuration.
10
11         Args:
12             origin (list or numpy.ndarray, optional): Origin point.
13                 Defaults to [0, 0, 0].
14             d (float, optional): Distance parameter. Defaults to 1.0.

```

```

15     theta (float, optional): Angle parameter in radians.
16         Defaults to pi/4.
17     plot_type (str, optional): Type of plot, either "3d" or "2d".
18         Defaults to "3d".
19     title (str, optional): Plot title. Defaults to None.
20     show_plot (bool, optional): Whether to show the plot.
21         Defaults to True.
22     save_path (str, optional): Path to save the plot. Defaults to None.
23     """

```

This class provides a unified way to configure all aspects of vector generation and visualization. It stores the configuration parameters and provides methods for saving configurations to and loading configurations from JSON files.

4.3.2 VectorConfig.save_to_file

```

1 def save_to_file(self, file_path):
2     """
3     Save the configuration to a JSON file.
4
5     Args:
6         file_path (str): Path to save the configuration file
7
8     Returns:
9         bool: True if successful, False otherwise
10    """

```

This method saves the configuration to a JSON file. It takes a file path as input and returns True if the save was successful and False otherwise.

4.3.3 VectorConfig.load_from_file

```

1 @classmethod
2 def load_from_file(cls, file_path):
3     """
4     Load the configuration from a JSON file.
5
6     Args:
7         file_path (str): Path to the configuration file
8
9     Returns:
10        VectorConfig: The loaded configuration
11    """

```

This class method loads a configuration from a JSON file. It takes a file path as input and returns a new `VectorConfig` object with the loaded configuration.

4.4 main Module

4.4.1 main Function

```

1 def main():
2     """
3     Main function for the command-line interface.
4     """

```

This function is the entry point for the command-line interface. It parses command-line arguments, creates a configuration, generates orthogonal vectors, and visualizes them.

4.5 __init__ Module

The `__init__.py` module exports the key functions and classes from the package, making them available when the package is imported:

```

1 from .vector_utils import create_orthogonal_vectors, check_orthogonality
2 from .visualization import plot_vectors, plot_vectors_3d, plot_vectors_2d
3 from .config import VectorConfig
4
5 __all__ = [

```

```

6     'create_orthogonal_vectors',
7     'check_orthogonality',
8     'plot_vectors',
9     'plot_vectors_3d',
10    'plot_vectors_2d',
11    'VectorConfig'
12 ]

```

5 Usage Examples

This section provides examples of how to use the Generalized Orthogonal Vectors Generator and Visualizer package. It includes examples of using the package as a Python module and as a command-line tool.

5.1 Basic Usage as a Python Module

The following example shows how to use the package as a Python module to generate and visualize orthogonal vectors with default parameters:

```

1 import numpy as np
2 from generalized import create_orthogonal_vectors, plot_vectors
3
4 # Generate orthogonal vectors with default parameters
5 # (origin at [0, 0, 0], d=1.0, theta=pi/4)
6 vectors = create_orthogonal_vectors(origin=[0, 0, 0])
7
8 # Plot the vectors in 3D
9 plot_vectors(vectors, origin=[0, 0, 0])

```

5.2 Customizing Vector Generation

The following example shows how to customize the vector generation by specifying the origin, distance parameter, and angle parameter:

```

1 import numpy as np
2 import math
3 from generalized import create_orthogonal_vectors, plot_vectors
4
5 # Generate orthogonal vectors with custom parameters
6 origin = [1, 1, 1]
7 d = 2.0
8 theta = math.pi / 3
9
10 vectors = create_orthogonal_vectors(origin=origin, d=d, theta=theta)
11
12 # Plot the vectors in 3D
13 plot_vectors(vectors, origin=origin, title=f"Orthogonal Vectors (Origin={origin}, d={d}, theta={theta})")

```

5.3 Using the VectorConfig Class

The following example shows how to use the `VectorConfig` class to configure vector generation and visualization:

```

1 import numpy as np
2 import math
3 from generalized import create_orthogonal_vectors, plot_vectors, VectorConfig
4
5 # Create a configuration
6 config = VectorConfig(
7     origin=[0, 0, 2],
8     d=1.5,
9     theta=math.pi / 6,
10    plot_type="2d",
11    title="Custom Configuration",
12    save_path="custom_config.png"
13 )

```

```

14
15 # Generate orthogonal vectors using the configuration
16 vectors = create_orthogonal_vectors(
17     origin=config.origin,
18     d=config.d,
19     theta=config.theta
20 )
21
22 # Plot the vectors using the configuration
23 plot_vectors(
24     vectors,
25     origin=config.origin,
26     plot_type=config.plot_type,
27     title=config.title,
28     show_plot=config.show_plot,
29     save_path=config.save_path
30 )

```

5.4 Saving and Loading Configurations

The following example shows how to save a configuration to a file and load it later:

```

1 import numpy as np
2 import math
3 from generalized import VectorConfig, create_orthogonal_vectors, plot_vectors
4
5 # Create a configuration
6 config = VectorConfig(
7     origin=[0, 0, 2],
8     d=1.5,
9     theta=math.pi / 6,
10    plot_type="2d",
11    title="Custom Configuration"
12 )
13
14 # Save the configuration to a file
15 config.save_to_file("config.json")
16
17 # Later, load the configuration from the file
18 loaded_config = VectorConfig.load_from_file("config.json")
19
20 # Generate orthogonal vectors using the loaded configuration
21 vectors = create_orthogonal_vectors(
22     origin=loaded_config.origin,
23     d=loaded_config.d,
24     theta=loaded_config.theta
25 )
26
27 # Plot the vectors using the loaded configuration
28 plot_vectors(
29     vectors,
30     origin=loaded_config.origin,
31     plot_type=loaded_config.plot_type,
32     title=loaded_config.title,
33     show_plot=loaded_config.show_plot,
34     save_path=loaded_config.save_path
35 )

```

5.5 Checking Orthogonality

The following example shows how to check if a set of vectors is orthogonal:

```

1 import numpy as np
2 from generalized import create_orthogonal_vectors, check_orthogonality
3
4 # Generate orthogonal vectors
5 vectors = create_orthogonal_vectors(origin=[0, 0, 0])
6
7 # Check if the vectors are orthogonal
8 is_orthogonal = check_orthogonality(vectors, origin=[0, 0, 0])
9
10 print(f"Vectors are orthogonal: {is_orthogonal}")

```

5.6 Using the Command-line Interface

The following examples show how to use the command-line interface to generate and visualize orthogonal vectors:

5.6.1 Basic Usage

```
1 python -m generalized.main
```

This command generates and visualizes orthogonal vectors with default parameters (origin at [0, 0, 0], d=1.0, theta=pi/4).

5.6.2 Customizing Vector Generation

```
1 python -m generalized.main --origin 1 1 1 --d 2.0 --theta 1.047
```

This command generates and visualizes orthogonal vectors with custom parameters (origin at [1, 1, 1], d=2.0, theta=pi/3).

5.6.3 Customizing Visualization

```
1 python -m generalized.main --plot-type 2d --title "Custom Visualization" --save-path
  custom.png
```

This command generates orthogonal vectors with default parameters and visualizes them with custom visualization options (2D plot, custom title, save to file).

5.6.4 Using a Configuration File

```
1 python -m generalized.main --config config.json
```

This command loads a configuration from a file and uses it to generate and visualize orthogonal vectors.

5.6.5 Saving a Configuration File

```
1 python -m generalized.main --origin 1 1 1 --d 2.0 --theta 1.047 --save-config config.json
```

This command generates and visualizes orthogonal vectors with custom parameters and saves the configuration to a file.

5.7 Complete Example Script

The following is a complete example script that demonstrates various features of the package:

```
1 import numpy as np
2 import math
3 import matplotlib.pyplot as plt
4 from generalized import create_orthogonal_vectors, check_orthogonality, plot_vectors,
  VectorConfig
5
6 def main():
7     # Create configurations for different examples
8     configs = [
9         VectorConfig(
10             origin=[0, 0, 0],
11             d=1.0,
12             theta=math.pi / 4,
13             plot_type="3d",
14             title="Default Configuration",
15             save_path="default.png"
16         ),
17         VectorConfig(
18             origin=[1, 1, 1],
19             d=2.0,
20             theta=math.pi / 3,
```

```

21         plot_type="3d",
22         title="Custom Configuration 1",
23         save_path="custom1.png"
24     ),
25     VectorConfig(
26         origin=[0, 0, 2],
27         d=1.5,
28         theta=math.pi / 6,
29         plot_type="2d",
30         title="Custom Configuration 2",
31         save_path="custom2.png"
32     )
33 ]
34
35 # Process each configuration
36 for i, config in enumerate(configs):
37     print(f"Processing configuration {i+1}/{len(configs)}")
38
39     # Generate orthogonal vectors
40     vectors = create_orthogonal_vectors(
41         origin=config.origin,
42         d=config.d,
43         theta=config.theta
44     )
45
46     # Check orthogonality
47     is_orthogonal = check_orthogonality(vectors, origin=config.origin)
48     print(f"  Vectors are orthogonal: {is_orthogonal}")
49
50     # Plot vectors
51     plot_vectors(
52         vectors,
53         origin=config.origin,
54         plot_type=config.plot_type,
55         title=config.title,
56         show_plot=False,
57         save_path=config.save_path
58     )
59     print(f"  Plot saved to {config.save_path}")
60
61     # Save configuration
62     config_file = f"config{i+1}.json"
63     config.save_to_file(config_file)
64     print(f"  Configuration saved to {config_file}")
65
66     # Show all plots
67     plt.show()
68
69 if __name__ == "__main__":
70     main()

```

This script creates three different configurations, generates orthogonal vectors for each, checks their orthogonality, plots them, and saves both the plots and configurations to files. Finally, it displays all the plots.

6 Visualization

The Generalized Orthogonal Vectors Generator and Visualizer package provides various visualization options for the generated vectors. This section describes the visualization techniques used by the package.

6.1 3D Visualization

The 3D visualization shows the vectors in three-dimensional space. It uses Matplotlib's 3D plotting capabilities to create a 3D plot with the following features:

- The origin point is shown as a black dot.
- The vectors are shown as arrows from the origin point.
- Each vector is assigned a different color for easy identification.

- The plot includes a legend identifying each vector.
- The plot includes labels for the X, Y, and Z axes.
- The plot includes a title, which can be customized.

The 3D visualization provides a complete view of the vectors in three-dimensional space, allowing for a better understanding of their spatial relationships.

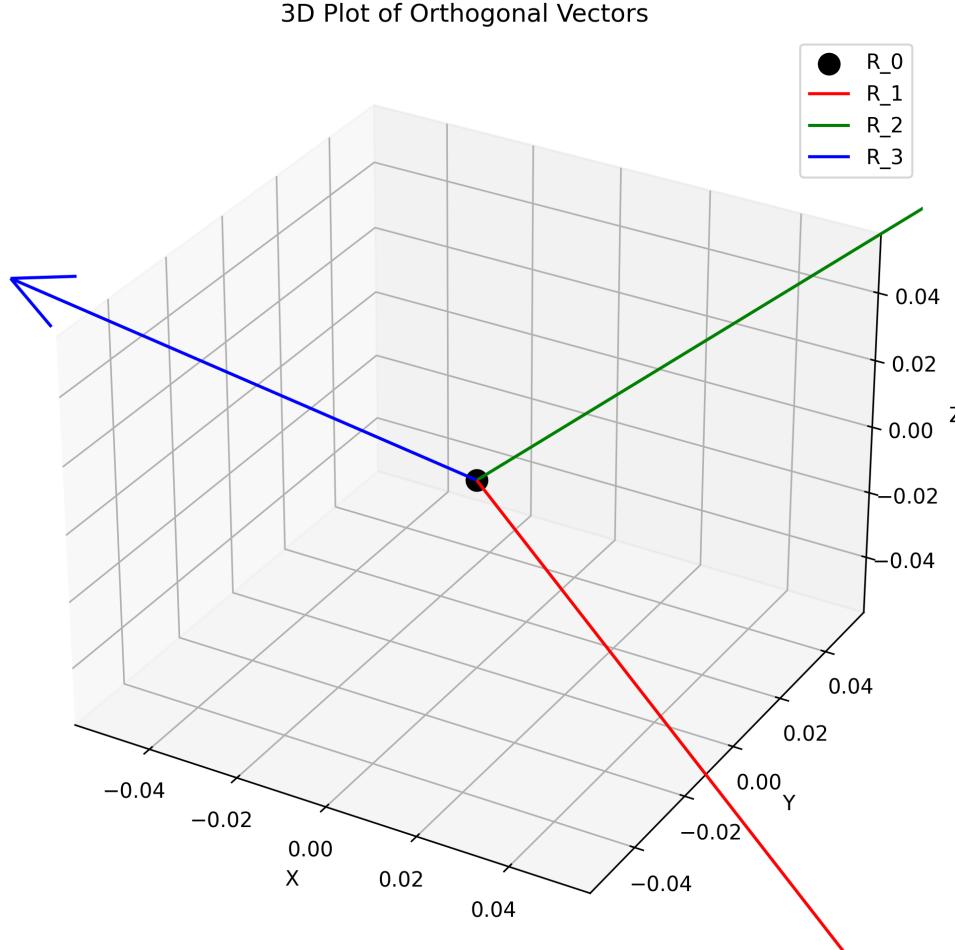


Figure 1: Example of 3D visualization

6.2 2D Projections

The 2D visualization shows projections of the vectors onto various planes. It creates four subplots showing the following projections:

- XY Plane: Shows the projection of the vectors onto the XY plane ($Z=0$).
- XZ Plane: Shows the projection of the vectors onto the XZ plane ($Y=0$).
- YZ Plane: Shows the projection of the vectors onto the YZ plane ($X=0$).
- R0 Plane: Shows the projection of the vectors onto the plane perpendicular to the vector from the global origin to R_0 . This provides a clear view of the orthogonality of the vectors in the plane normal to the origin direction. The projection uses two orthogonal basis vectors in this plane:
 - If R_0 is the origin $(0,0,0)$, the basis vectors are derived from R_1 and R_2 after orthogonalization.
 - Otherwise, the first basis vector is computed as the cross product of the normalized R_0 vector with either $[1,0,0]$ or $[0,1,0]$ (whichever is not parallel to R_0).

- The second basis vector is computed as the cross product of the normalized R_0 vector with the first basis vector, ensuring a perfectly orthogonal basis.

Each subplot includes the following features:

- The origin point is shown as a black dot.
- The vectors are shown as arrows from the origin point.
- Each vector is assigned a different color for easy identification.
- The subplot includes a legend identifying each vector.
- The subplot includes labels for the axes.
- The subplot includes a title indicating the plane of projection.

The 2D projections provide different perspectives on the vectors, allowing for a better understanding of their projections onto different planes.

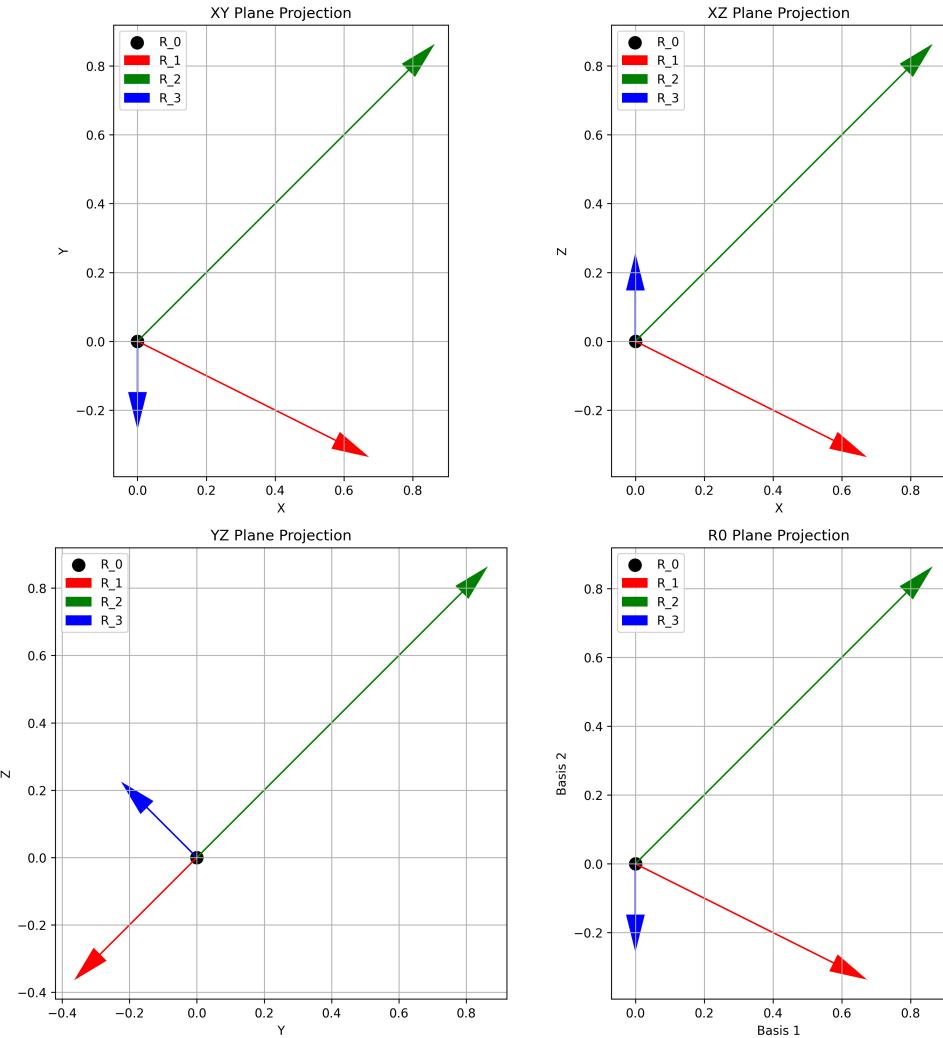


Figure 2: Example of 2D projections

6.3 Customization Options

The visualization functions provide various customization options, including:

- Plot title: The title of the plot can be customized.

- Show plot: The plot can be displayed interactively or not.
- Save path: The plot can be saved to a file instead of being displayed interactively.

These options can be specified directly when calling the visualization functions or through the `VectorConfig` class.

6.4 R0 Plane Projections

The package includes specialized scripts for generating R0 plane projections, which provide a clear view of the orthogonality of the vectors in the plane perpendicular to the origin direction:

- `generate_r0_projections.py`: Generates R0 plane projections for various configurations with improved axis handling, symmetric axis limits, and better legend placement.
- `generate_combined_views.py`: Generates combined 3D and R0 plane projection figures, showing both perspectives side by side.
- `generate_specific_r0_projections.py`: Generates R0 plane projections specifically for the three combined effect configurations (origins at $(0, 0, 0)$, $(1, 1, 1)$, and $(0, 0, 2)$).

These projections include the following enhancements:

- Dynamic axis limit calculation with a 20% margin to ensure all vector projections are visible
- Symmetric axis limits for better visual representation
- Improved grid for better readability
- Enhanced legend placement

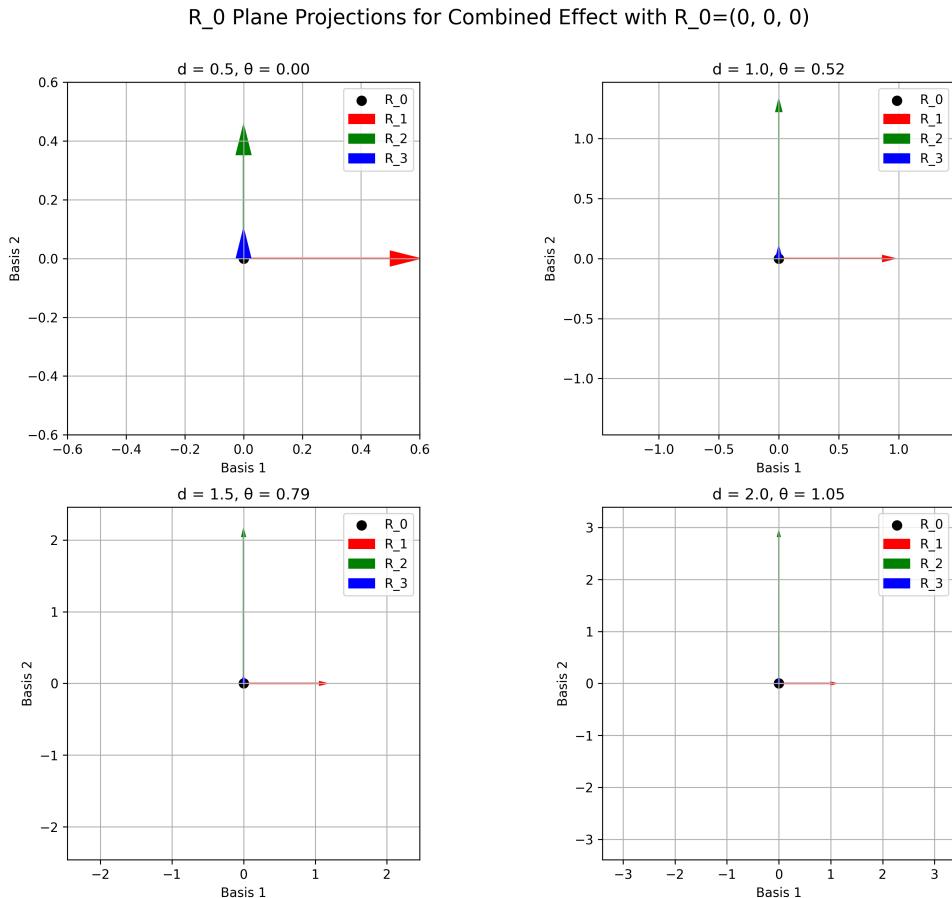


Figure 3: Example of R0 plane projections

6.5 Implementation Details

The visualization functions use Matplotlib to create the plots. The 3D visualization uses Matplotlib's `Axes3D` class, while the 2D visualizations use regular Matplotlib axes.

The vectors are plotted using Matplotlib's `quiver` function, which creates arrows from a starting point to an ending point. The origin point is plotted using Matplotlib's `scatter` function.

The colors of the vectors are assigned using Matplotlib's default color cycle, ensuring that each vector has a different color.

The legends are created using Matplotlib's `legend` function, with labels for each vector.

The plots are saved using Matplotlib's `savefig` function, which supports various file formats, including PNG, JPEG, and PDF.

6.6 Visualization in the Command-line Interface

The command-line interface provides options for controlling the visualization, including:

- `--plot-type`: Specifies the type of plot, either "3d" or "2d".
- `--title`: Specifies the title of the plot.
- `--no-show`: Prevents the plot from being displayed interactively.
- `--save-path`: Specifies the path to save the plot.

These options allow users to customize the visualization without modifying the code.

7 Configuration

The Generalized Orthogonal Vectors Generator and Visualizer package provides a flexible configuration system that allows users to customize all aspects of vector generation and visualization. This section describes the configuration system and its features.

7.1 VectorConfig Class

The configuration system is implemented through the `VectorConfig` class, which provides a unified way to configure all aspects of vector generation and visualization. The class has the following attributes:

- `origin`: The origin point for vector generation (default: [0, 0, 0]).
- `d`: The distance parameter for vector generation (default: 1.0).
- `theta`: The angle parameter for vector generation (default: $\pi/4$).
- `plot_type`: The type of plot, either "3d" or "2d" (default: "3d").
- `title`: The title of the plot (default: None, which uses a default title based on the plot type).
- `show_plot`: Whether to show the plot interactively (default: True).
- `save_path`: The path to save the plot (default: None, which doesn't save the plot).

The class provides methods for initializing the configuration, saving it to a file, and loading it from a file.

7.2 Initializing a Configuration

A configuration can be initialized with default values or with custom values:

```

1 # Initialize with default values
2 config = VectorConfig()
3
4 # Initialize with custom values
5 config = VectorConfig(
6     origin=[1, 1, 1],
7     d=2.0,
8     theta=math.pi / 3,
9     plot_type="2d",
10    title="Custom Configuration",
11    show_plot=False,
12    save_path="custom.png"
13 )

```

7.3 Using a Configuration

A configuration can be used to generate and visualize orthogonal vectors:

```

1 # Generate orthogonal vectors using the configuration
2 vectors = create_orthogonal_vectors(
3     origin=config.origin,
4     d=config.d,
5     theta=config.theta
6 )
7
8 # Plot the vectors using the configuration
9 plot_vectors(
10    vectors,
11    origin=config.origin,
12    plot_type=config.plot_type,
13    title=config.title,
14    show_plot=config.show_plot,
15    save_path=config.save_path
16 )

```

7.4 Saving a Configuration to a File

A configuration can be saved to a JSON file for later use:

```

1 # Save the configuration to a file
2 config.save_to_file("config.json")

```

The saved file will contain all the configuration parameters in JSON format:

```

1 {
2     "origin": [1, 1, 1],
3     "d": 2.0,
4     "theta": 1.0471975511965976,
5     "plot_type": "2d",
6     "title": "Custom Configuration",
7     "show_plot": false,
8     "save_path": "custom.png"
9 }

```

7.5 Loading a Configuration from a File

A configuration can be loaded from a JSON file:

```

1 # Load the configuration from a file
2 config = VectorConfig.load_from_file("config.json")

```

This creates a new `VectorConfig` object with the parameters specified in the file.

7.6 Configuration in the Command-line Interface

The command-line interface provides options for configuring vector generation and visualization:

- `--origin`: Specifies the origin point as three space-separated values.
- `--d`: Specifies the distance parameter.
- `--theta`: Specifies the angle parameter in radians.
- `--plot-type`: Specifies the type of plot, either "3d" or "2d".
- `--title`: Specifies the title of the plot.
- `--no-show`: Prevents the plot from being displayed interactively.
- `--save-path`: Specifies the path to save the plot.
- `--config`: Specifies a configuration file to load.
- `--save-config`: Specifies a file to save the configuration to.

These options allow users to customize the configuration without modifying the code.

7.7 Configuration File Format

The configuration file is a JSON file with the following structure:

```

1 {
2     "origin": [x, y, z],
3     "d": float,
4     "theta": float,
5     "plot_type": "3d" or "2d",
6     "title": string or null,
7     "show_plot": boolean,
8     "save_path": string or null
9 }
```

All fields are optional and will use default values if not specified.

7.8 Default Configuration

The default configuration is as follows:

- `origin`: [0, 0, 0]
- `d`: 1.0
- `theta`: $\pi/4$ (approximately 0.7853981633974483)
- `plot_type`: "3d"
- `title`: None (uses a default title based on the plot type)
- `show_plot`: True
- `save_path`: None (doesn't save the plot)

This configuration generates three orthogonal vectors from the origin [0, 0, 0] with a distance parameter of 1.0 and an angle parameter of $\pi/4$, and visualizes them in 3D.

8 Command-line Interface

The Generalized Orthogonal Vectors Generator and Visualizer package provides a command-line interface that allows users to generate and visualize orthogonal vectors without writing Python code. This section describes the command-line interface and its features.

8.1 Basic Usage

The command-line interface can be accessed by running the `main.py` module:

```
1 python -m generalized.main
```

This command generates and visualizes orthogonal vectors with default parameters (origin at [0, 0], d=1.0, theta=pi/4).

You can also access detailed help information by running:

```
1 python -m generalized.main help
```

This will display comprehensive information about all available commands, parameters, and usage examples.

8.2 Command-line Options

The command-line interface provides various options for configuring vector generation and visualization:

- `--origin X Y Z, -R X Y Z`: Specifies the origin point as three space-separated values. Default: 0 0 0.
- `--distance D, -d D`: Specifies the distance parameter. Default: 1.0.
- `--angle THETA, -a THETA`: Specifies the angle parameter in radians. Default: 0.7853981633974483 (pi/4).
- `--plot-type TYPE`: Specifies the type of plot, either "3d" or "2d". Default: "3d".
- `--title TITLE`: Specifies the title of the plot. Default: None (uses a default title based on the plot type).
- `--no-show`: Prevents the plot from being displayed interactively. Default: False (shows the plot).
- `--save-path PATH`: Specifies the path to save the plot. Default: None (doesn't save the plot).
- `--config FILE`: Specifies a configuration file to load. Default: None (uses command-line options).
- `--save-config FILE`: Specifies a file to save the configuration to. Default: None (doesn't save the configuration).
- `--check-orthogonality`: Checks if the generated vectors are orthogonal and prints the result. Default: False (doesn't check).
- `--verbose`: Enables verbose output, including vector coordinates and dot products. Default: False (minimal output).
- `--help`: Shows the help message and exits.

8.3 Examples

8.3.1 Customizing Vector Generation

```
1 python -m generalized.main -R 1 1 1 -d 2.0 -a 1.047
```

This command generates and visualizes orthogonal vectors with custom parameters (origin at [1, 1, 1], d=2.0, theta=pi/3). Note the use of shorter flags (-R, -d, -a) for more concise commands.

8.3.2 Customizing Visualization

```
1 python -m generalized.main --plot-type 2d --title "Custom Visualization" --save-path custom.png
```

This command generates orthogonal vectors with default parameters and visualizes them with custom visualization options (2D plot, custom title, save to file).

8.3.3 Using a Configuration File

```
1 python -m generalized.main --config config.json
```

This command loads a configuration from a file and uses it to generate and visualize orthogonal vectors.

8.3.4 Saving a Configuration File

```
1 python -m generalized.main --origin 1 1 1 --d 2.0 --theta 1.047 --save-config config.json
```

This command generates and visualizes orthogonal vectors with custom parameters and saves the configuration to a file.

8.3.5 Checking Orthogonality

```
1 python -m generalized.main --check-orthogonality
```

This command generates orthogonal vectors with default parameters, visualizes them, and checks if they are orthogonal.

8.3.6 Verbose Output

```
1 python -m generalized.main --verbose
```

This command generates orthogonal vectors with default parameters, visualizes them, and prints verbose output, including vector coordinates and dot products.

8.4 Implementation Details

The command-line interface is implemented in the `main.py` module using the `argparse` module from the Python standard library. The module defines a `main` function that parses command-line arguments, creates a configuration, generates orthogonal vectors, and visualizes them.

The command-line interface follows these steps:

1. Parse command-line arguments using `argparse`.
2. If a configuration file is specified, load the configuration from the file.
3. Override the configuration with any command-line options that are specified.
4. Generate orthogonal vectors using the configuration.
5. If requested, check if the vectors are orthogonal and print the result.
6. If verbose output is enabled, print vector coordinates and dot products.
7. Visualize the vectors using the configuration.
8. If requested, save the configuration to a file.

8.5 Error Handling

The command-line interface includes error handling for various scenarios, including:

- Invalid command-line arguments (e.g., non-numeric values for numeric options).
- Invalid configuration file (e.g., file not found, invalid JSON).
- Invalid configuration parameters (e.g., negative distance parameter).

When an error occurs, the command-line interface prints an error message and exits with a non-zero exit code.

8.6 Help Message

The command-line interface provides a help message that can be displayed using the `--help` option:

```
1 python -m generalized.main --help
```

The help message includes a description of the program, a list of all available options, and examples of how to use the program.

9 Example Results

This section presents example results for different configurations of the Generalized Orthogonal Vectors Generator and Visualizer. It shows the generated vectors and their visualizations for various parameter values.

9.1 Default Configuration

The default configuration generates three orthogonal vectors from the origin [0, 0, 0] with a distance parameter of 1.0 and an angle parameter of $\pi/4$.

9.1.1 Vector Coordinates

The coordinates of the generated vectors are:

$$\vec{R}_1 = \begin{pmatrix} 0.8165 \\ -0.4082 \\ -0.4082 \end{pmatrix} \quad (17)$$

$$\vec{R}_2 = \begin{pmatrix} 0.5774 \\ 0.5774 \\ 0.5774 \end{pmatrix} \quad (18)$$

$$\vec{R}_3 = \begin{pmatrix} 0 \\ -0.7071 \\ 0.7071 \end{pmatrix} \quad (19)$$

9.1.2 Dot Products

The dot products between the displacement vectors are:

$$\vec{v}_1 \cdot \vec{v}_2 = 0 \quad (20)$$

$$\vec{v}_1 \cdot \vec{v}_3 = 0 \quad (21)$$

$$\vec{v}_2 \cdot \vec{v}_3 = 0 \quad (22)$$

These dot products confirm that the vectors are orthogonal.

9.1.3 3D Visualization

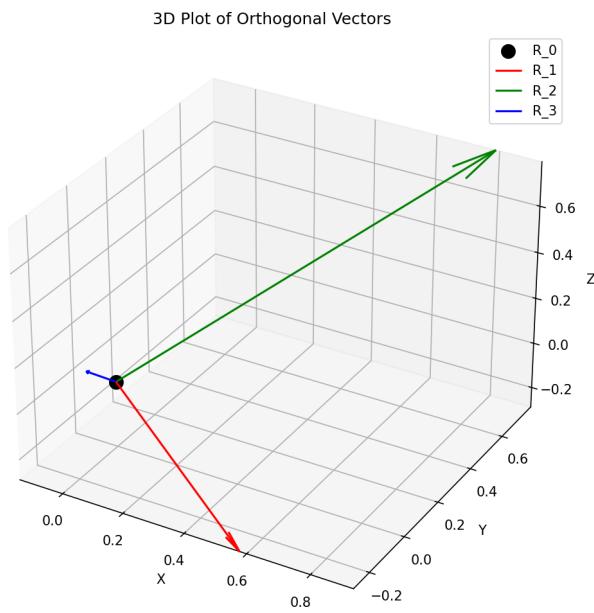


Figure 4: 3D visualization of the default configuration

9.1.4 2D Projections

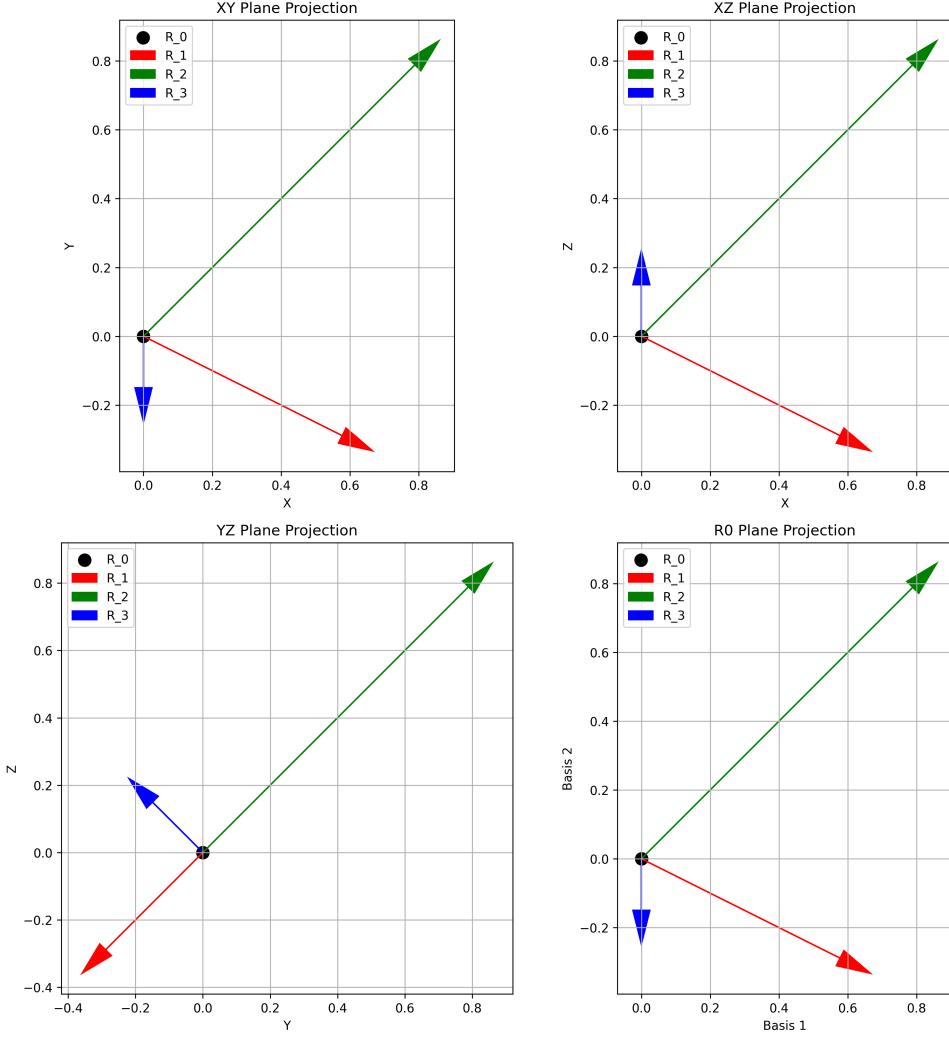


Figure 5: 2D projections of the default configuration

9.2 Custom Configuration 1

This configuration generates three orthogonal vectors from the origin [1, 1, 1] with a distance parameter of 2.0 and an angle parameter of $\pi/3$.

9.2.1 Vector Coordinates

The coordinates of the generated vectors are:

$$\vec{R}_1 = \begin{pmatrix} 2.6330 \\ 0.3165 \\ 0.3165 \end{pmatrix} \quad (23)$$

$$\vec{R}_2 = \begin{pmatrix} 1.6667 \\ 1.6667 \\ 1.6667 \end{pmatrix} \quad (24)$$

$$\vec{R}_3 = \begin{pmatrix} 1 \\ -0.1547 \\ 2.1547 \end{pmatrix} \quad (25)$$

9.2.2 Dot Products

The dot products between the displacement vectors are:

$$\vec{v}_1 \cdot \vec{v}_2 = 0 \quad (26)$$

$$\vec{v}_1 \cdot \vec{v}_3 = 0 \quad (27)$$

$$\vec{v}_2 \cdot \vec{v}_3 = 0 \quad (28)$$

These dot products confirm that the vectors are orthogonal.

9.2.3 3D Visualization

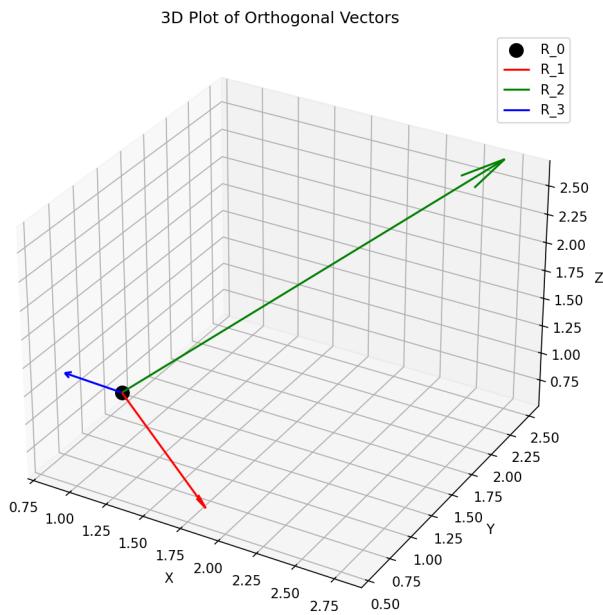


Figure 6: 3D visualization of custom configuration 1

9.2.4 2D Projections

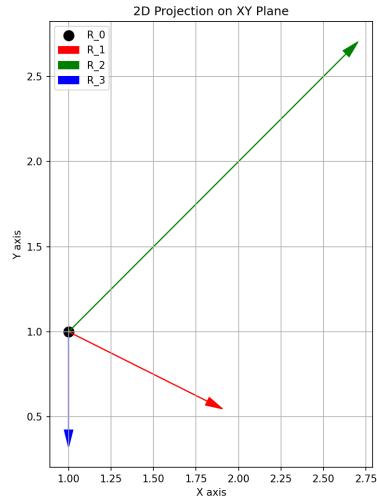


Figure 7: *
XY Projection

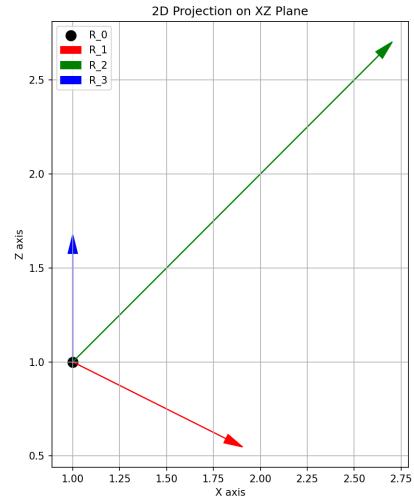


Figure 8: *
XZ Projection

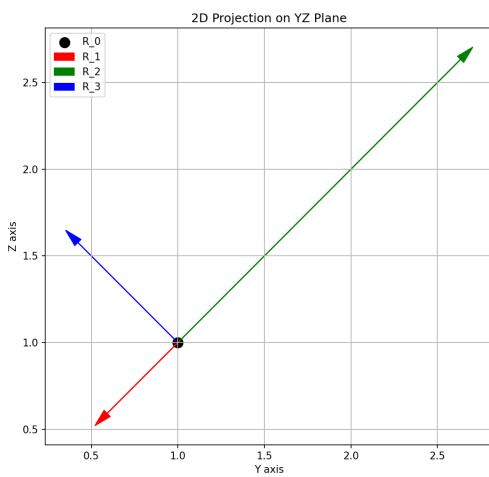


Figure 9: *
YZ Projection

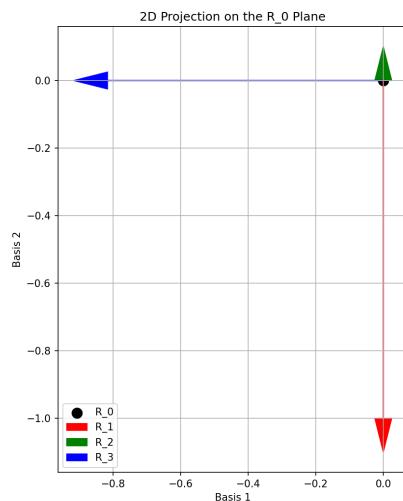


Figure 10: *
R0 Projection

Figure 11: 2D projections of custom configuration 1

9.3 Custom Configuration 2

This configuration generates three orthogonal vectors from the origin $[0, 0, 2]$ with a distance parameter of 1.5 and an angle parameter of $\pi/6$.

9.3.1 Vector Coordinates

The coordinates of the generated vectors are:

$$\vec{R}_1 = \begin{pmatrix} 1.2990 \\ -0.6495 \\ 1.3505 \end{pmatrix} \quad (29)$$

$$\vec{R}_2 = \begin{pmatrix} 0.7217 \\ 0.7217 \\ 2.7217 \end{pmatrix} \quad (30)$$

$$\vec{R}_3 = \begin{pmatrix} 0 \\ -0.3750 \\ 2.3750 \end{pmatrix} \quad (31)$$

9.3.2 Dot Products

The dot products between the displacement vectors are:

$$\vec{v}_1 \cdot \vec{v}_2 = 0 \quad (32)$$

$$\vec{v}_1 \cdot \vec{v}_3 = 0 \quad (33)$$

$$\vec{v}_2 \cdot \vec{v}_3 = 0 \quad (34)$$

These dot products confirm that the vectors are orthogonal.

9.3.3 3D Visualization

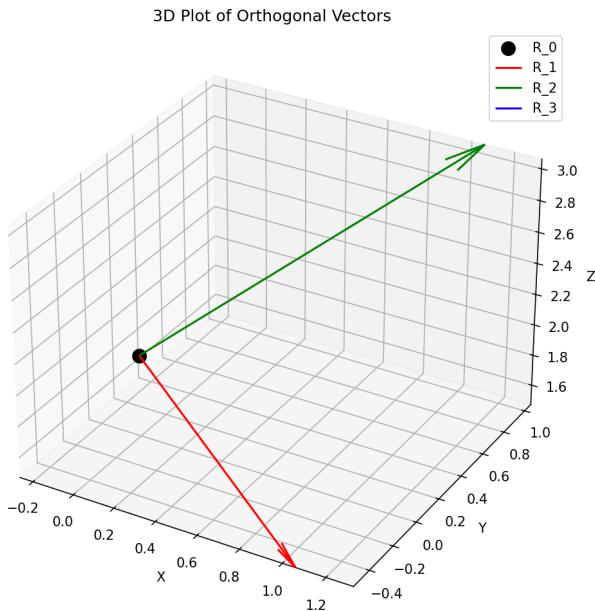


Figure 12: 3D visualization of custom configuration 2

9.3.4 2D Projections

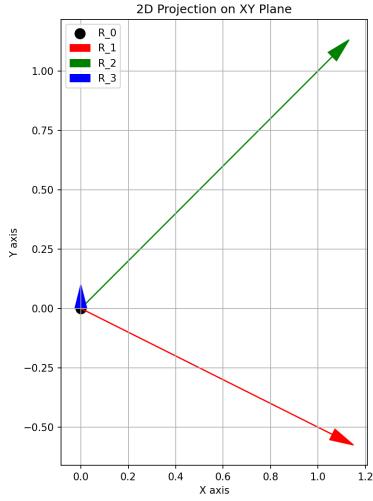


Figure 13: *
XY Projection

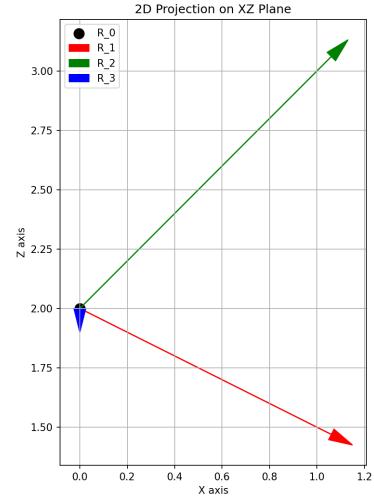


Figure 14: *
XZ Projection

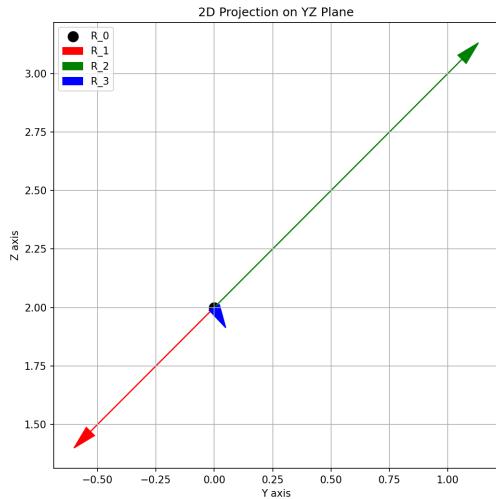


Figure 15: *
YZ Projection

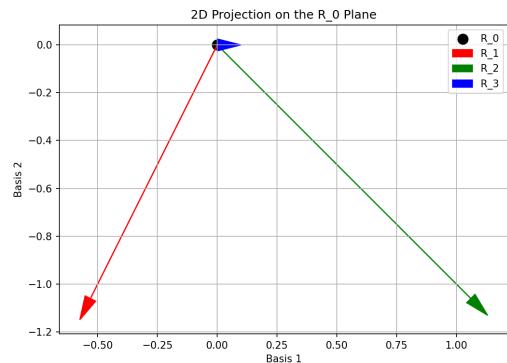


Figure 16: *
R0 Projection

Figure 17: 2D projections of custom configuration 2

9.4 Effect of Distance Parameter

The distance parameter d scales the vectors equally, preserving their orthogonality. Increasing d increases the distance of the vectors from the origin, while decreasing d decreases the distance.

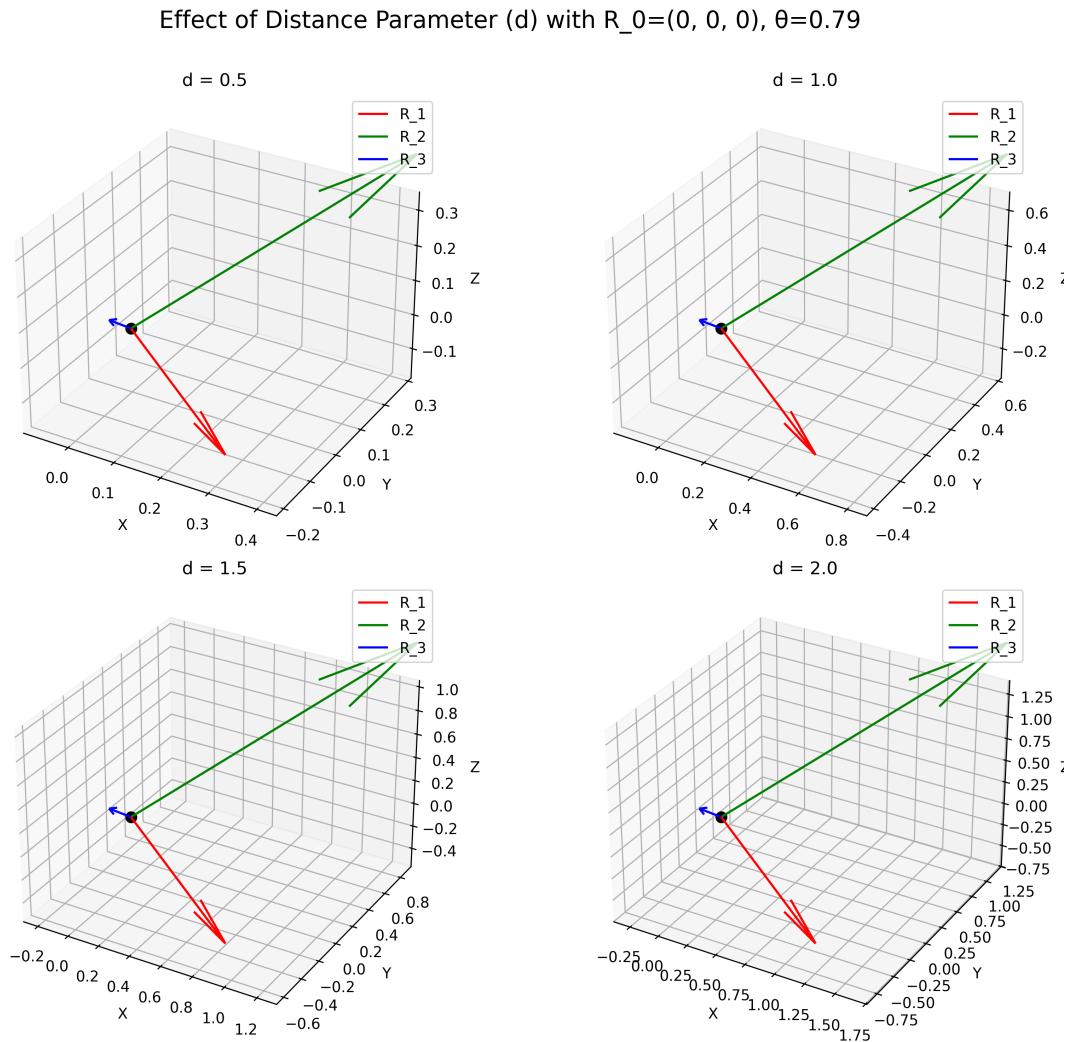
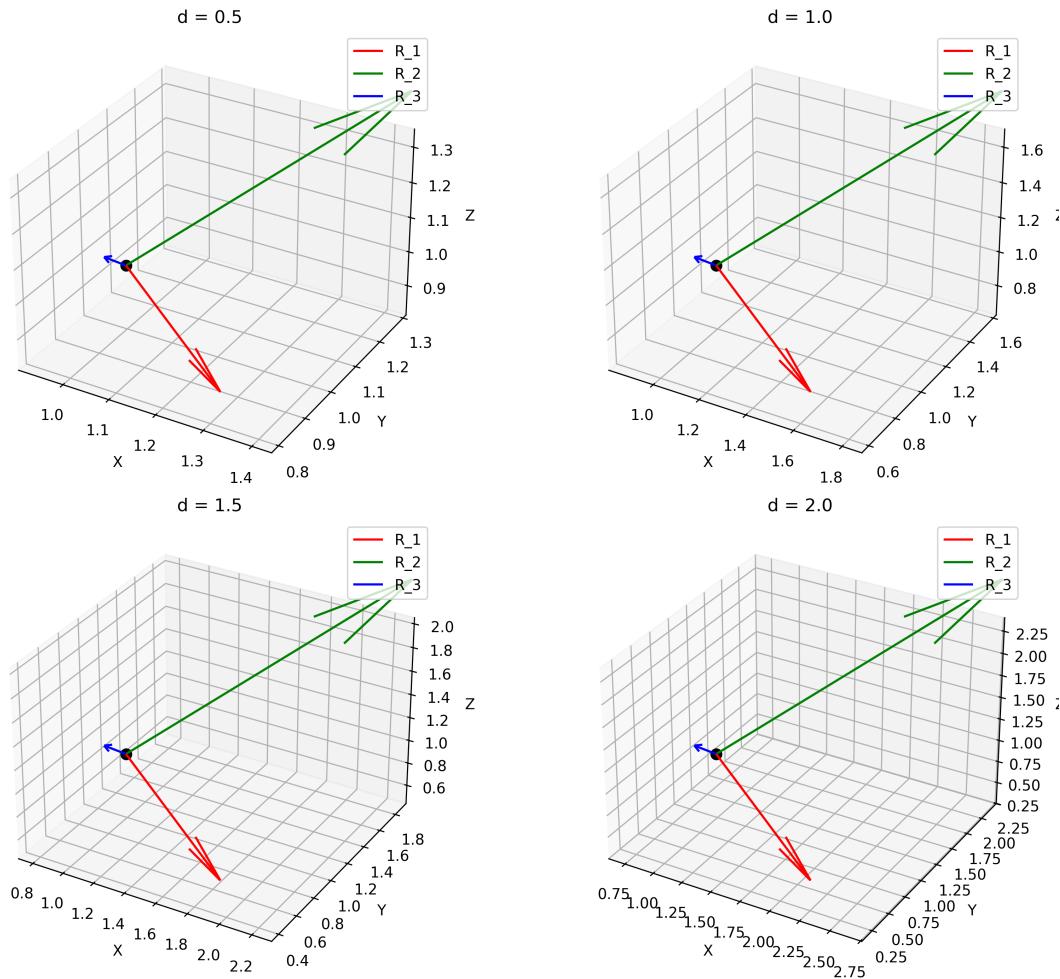


Figure 18: Effect of distance parameter on vector visualization with origin at $(0,0,0)$

Effect of Distance Parameter (d) with $R_0=(1, 1, 1)$, $\theta=0.79$ Figure 19: Effect of distance parameter on vector visualization with origin at $(1, 1, 1)$

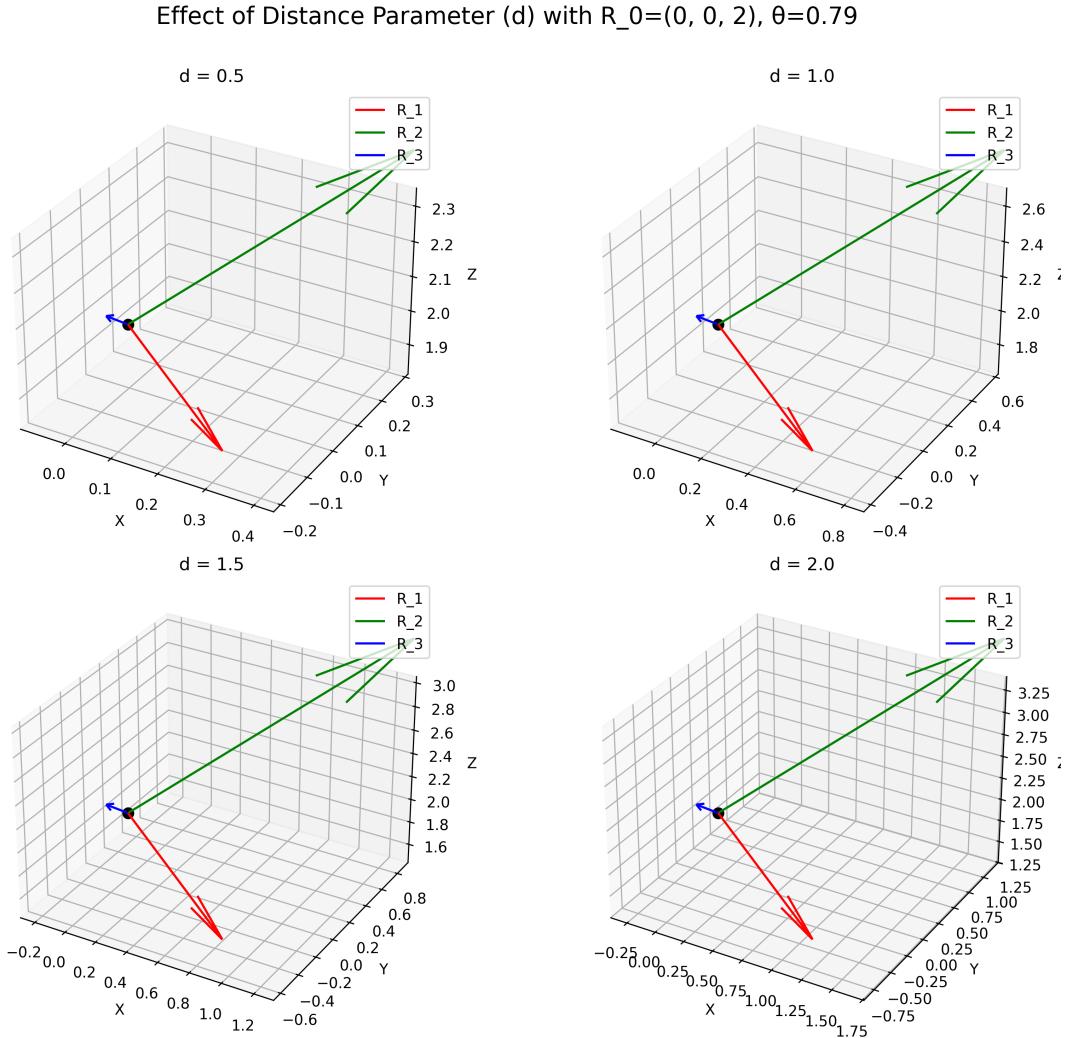
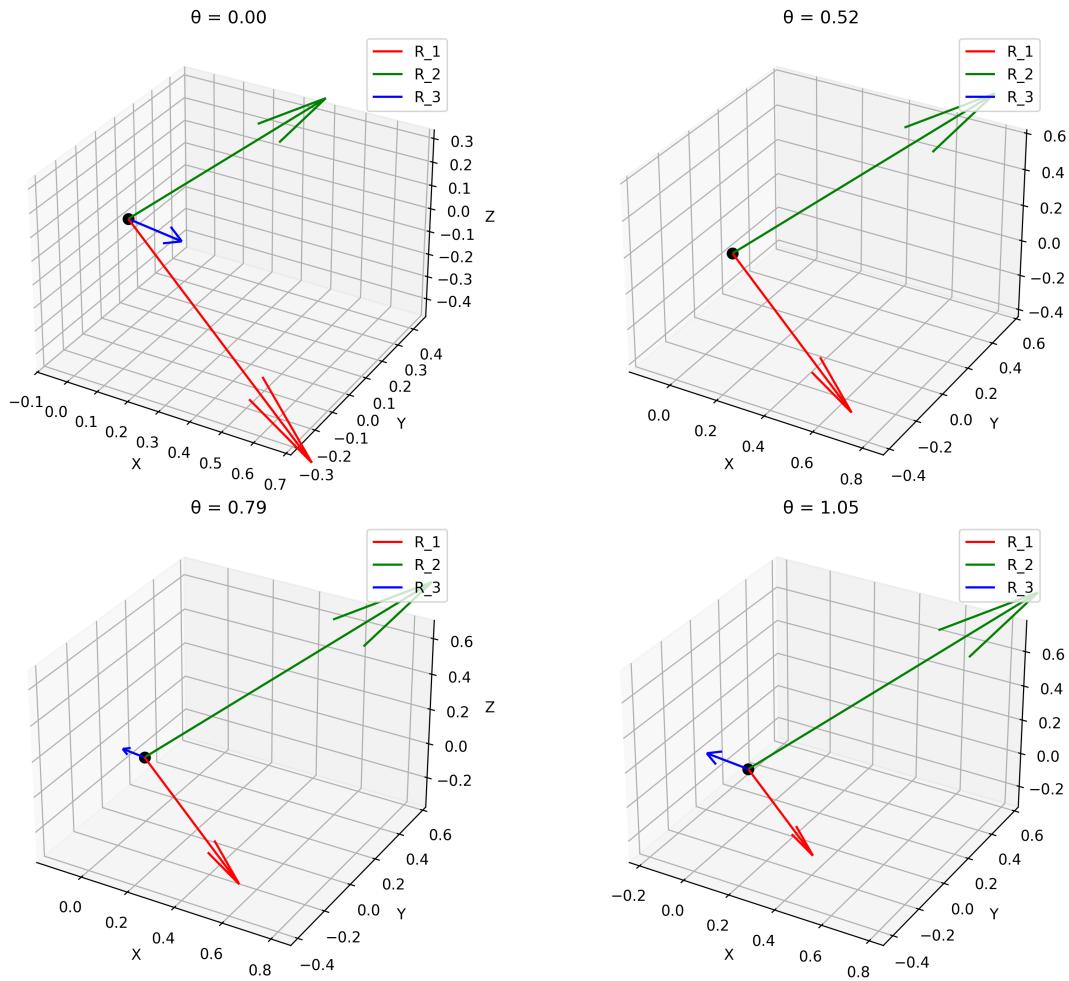


Figure 20: Effect of distance parameter on vector visualization with origin at $(0, 0, 2)$

Effect of Distance Parameter: The distance parameter d scales the vectors equally, preserving their orthogonality. Increasing d increases the distance of the vectors from the origin, while decreasing d decreases the distance. As shown in the figures above, the effect is consistent across different origin points.

9.5 Effect of Angle Parameter

The angle parameter θ rotates the vectors around the origin, preserving their orthogonality. Different values of θ result in different orientations of the vectors.

Effect of Angle Parameter (θ) with $R_0=(0, 0, 0)$, $d=1.0$ Figure 21: Effect of angle parameter on vector visualization with origin at $(0, 0, 0)$

Effect of Angle Parameter (θ) with $R_0=(1, 1, 1)$, $d=1.0$

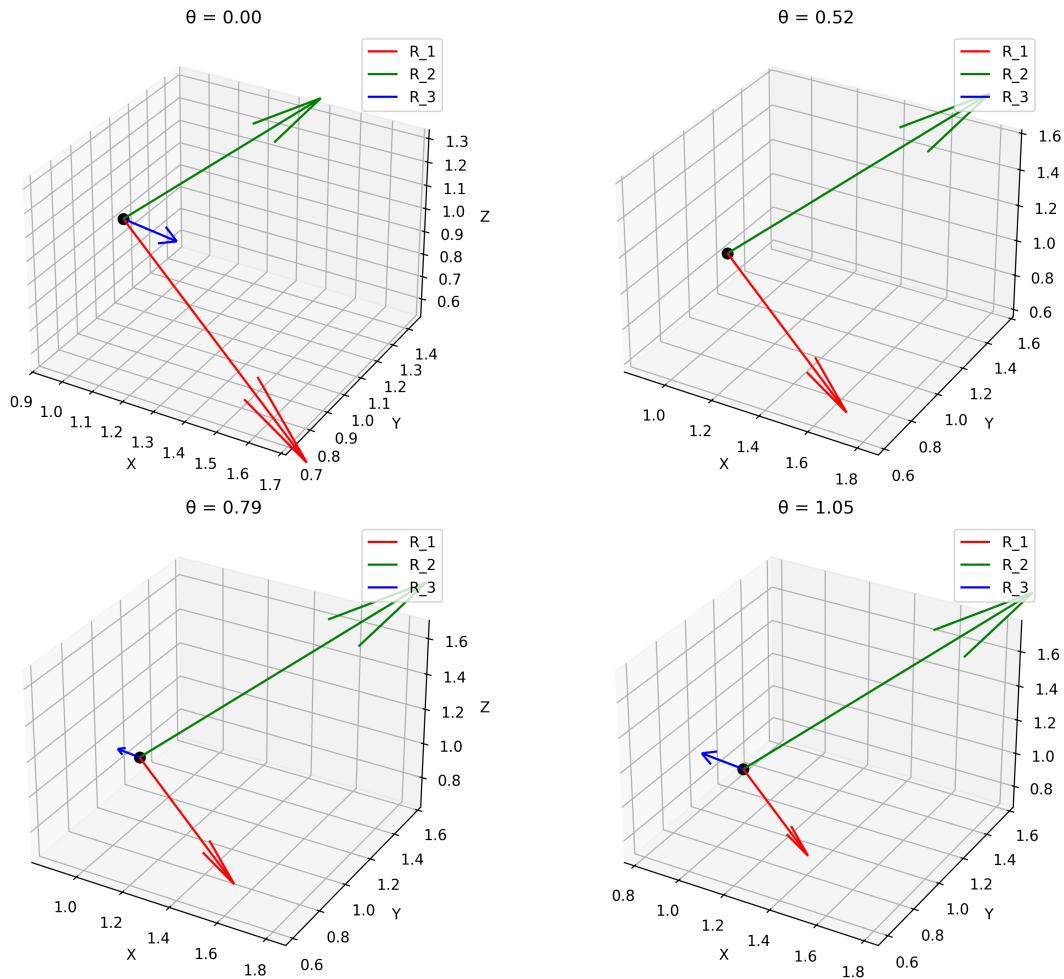
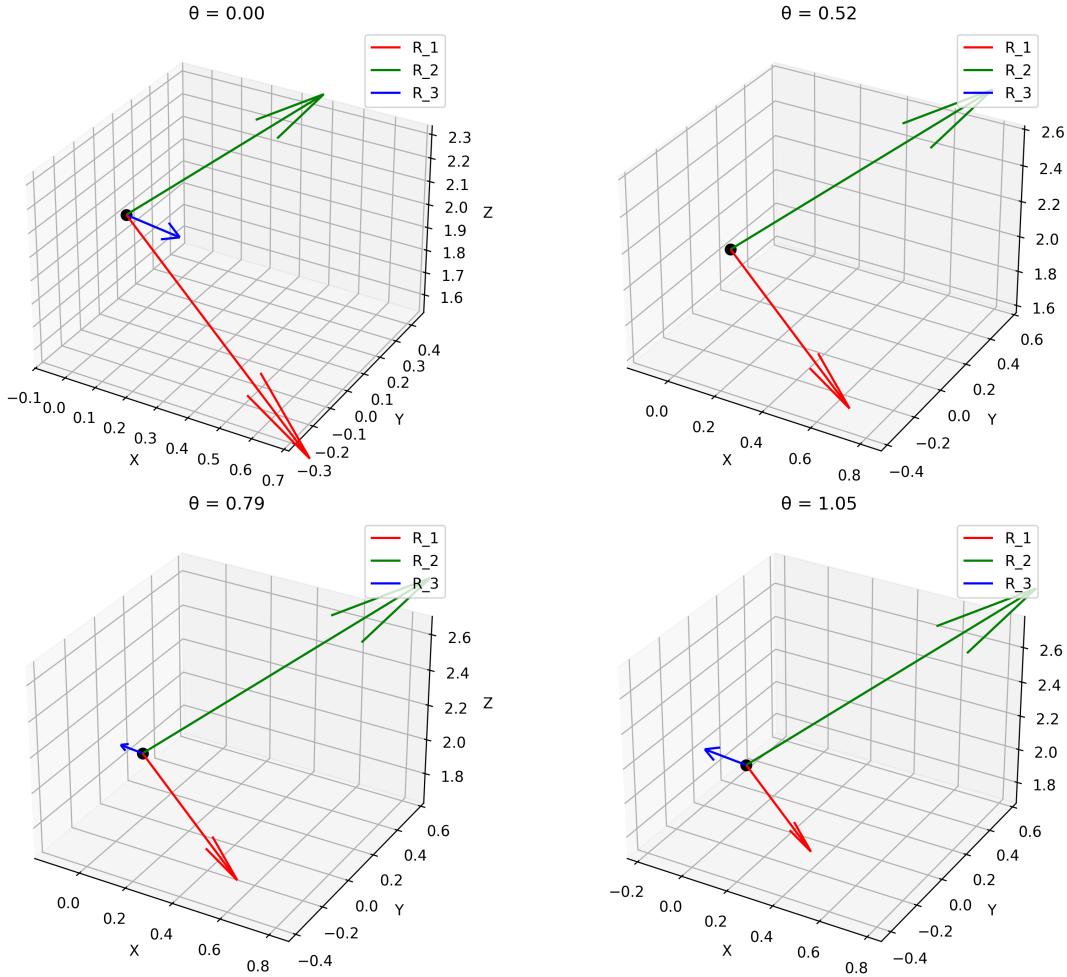


Figure 22: Effect of angle parameter on vector visualization with origin at $(1, 1, 1)$

Effect of Angle Parameter (θ) with $R_0=(0, 0, 2)$, $d=1.0$ Figure 23: Effect of angle parameter on vector visualization with origin at $(0, 0, 2)$

Effect of Angle Parameter: The angle parameter θ rotates the vectors around the origin, preserving their orthogonality. Different values of θ result in different orientations of the vectors. As shown in the figures above, the effect is consistent across different origin points, with the rotation occurring relative to the specified origin.

9.6 Effect of Origin

The origin parameter \vec{R}_0 shifts the entire vector system, preserving the orthogonality of the displacement vectors. Different origin points result in different positions of the vectors in space.

9.6.1 3D Visualizations

Combined Effect of Distance and Angle Parameters with $R_0=(0, 0, 0)$

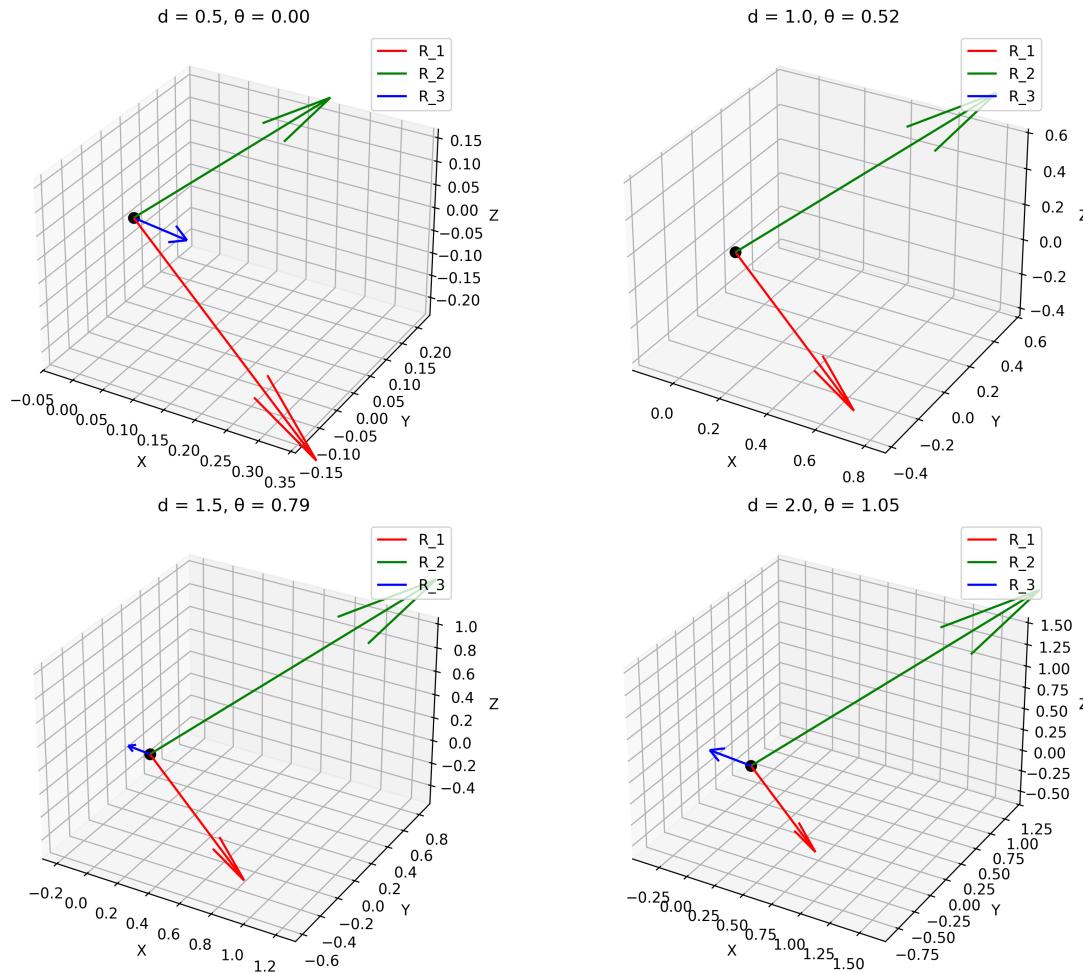
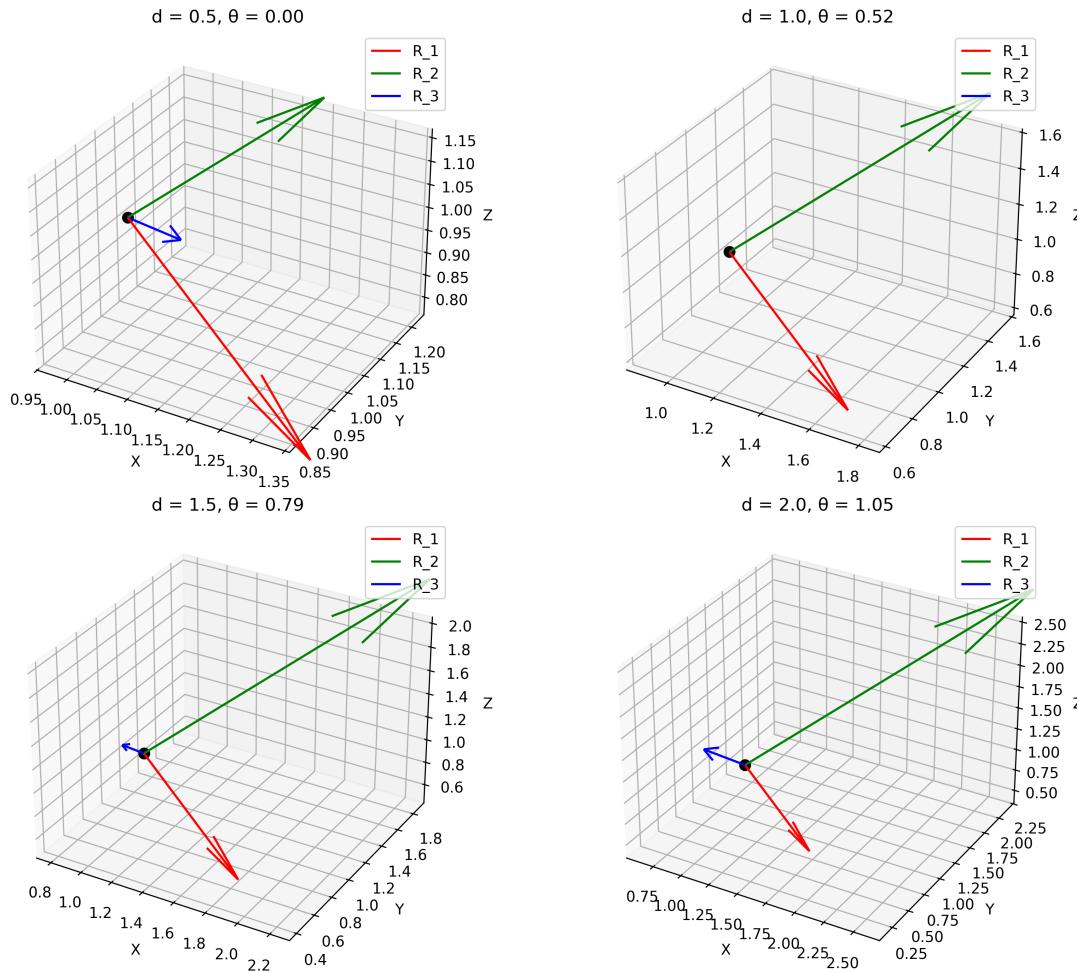


Figure 24: Combined effect of distance and angle parameters with origin at $(0, 0, 0)$

Combined Effect of Distance and Angle Parameters with $R_0 = (1, 1, 1)$ Figure 25: Combined effect of distance and angle parameters with origin at $(1, 1, 1)$

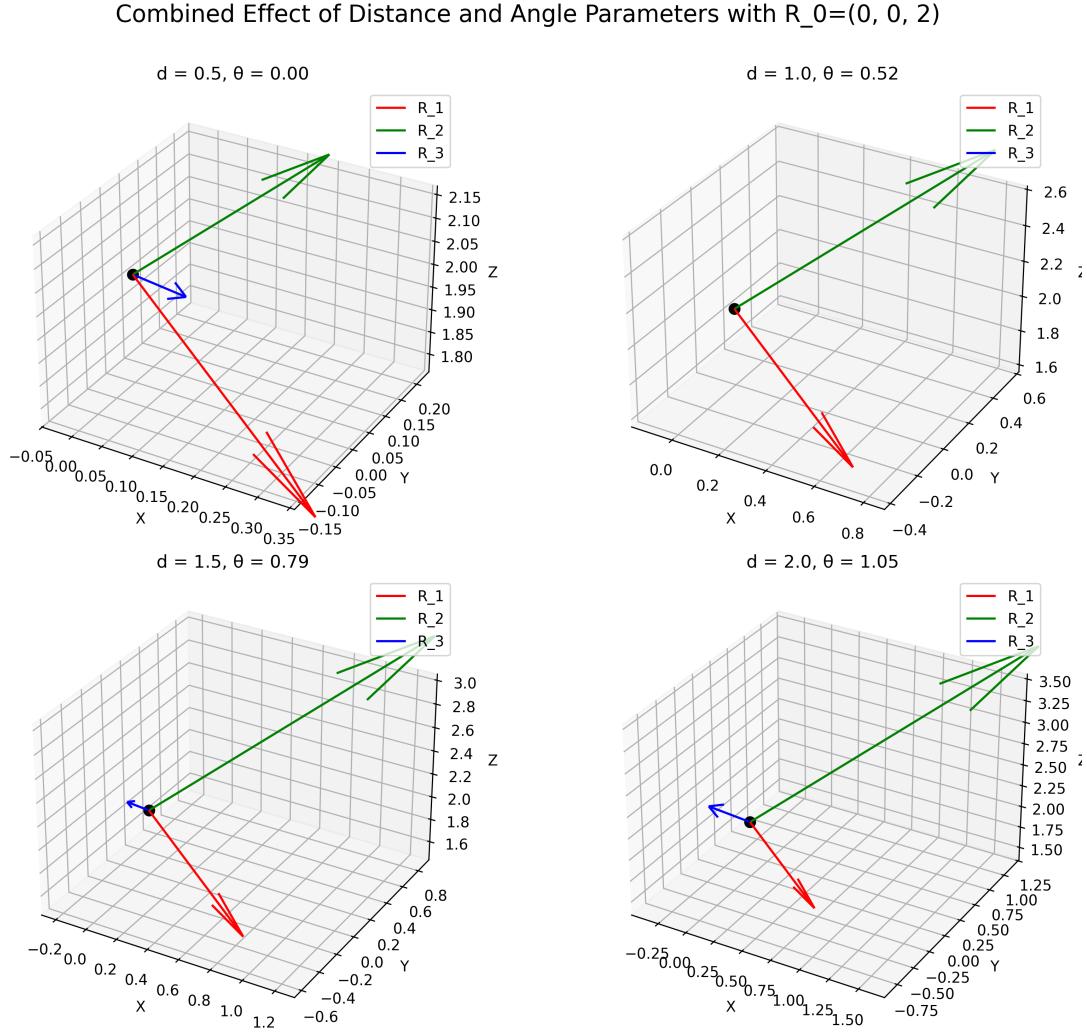


Figure 26: Combined effect of distance and angle parameters with origin at $(0, 0, 2)$

9.6.2 R_0 Plane Projections

The R_0 plane projections provide a clear view of the orthogonality of the vectors in the plane perpendicular to the origin direction. These projections use a carefully constructed orthogonal basis in the plane:

- When $R_0 = (0, 0, 0)$, the basis vectors are derived from R_1 and R_2 after orthogonalization using the Gram-Schmidt process.
- For non-zero R_0 , the basis is constructed as follows:
 - The plane normal is defined as the normalized vector from the global origin to R_0 .
 - The first basis vector is computed as the cross product of this normal with either $[1, 0, 0]$ or $[0, 1, 0]$ (whichever is not parallel to the normal).
 - The second basis vector is computed as the cross product of the normal with the first basis vector.
- This construction ensures that both basis vectors are orthogonal to each other and lie in the plane perpendicular to the vector from the origin to R_0 .

The projections use these basis vectors as the coordinate axes, with the origin of the projection being R_0 . This approach provides an optimal view of the orthogonality relationships between the vectors.

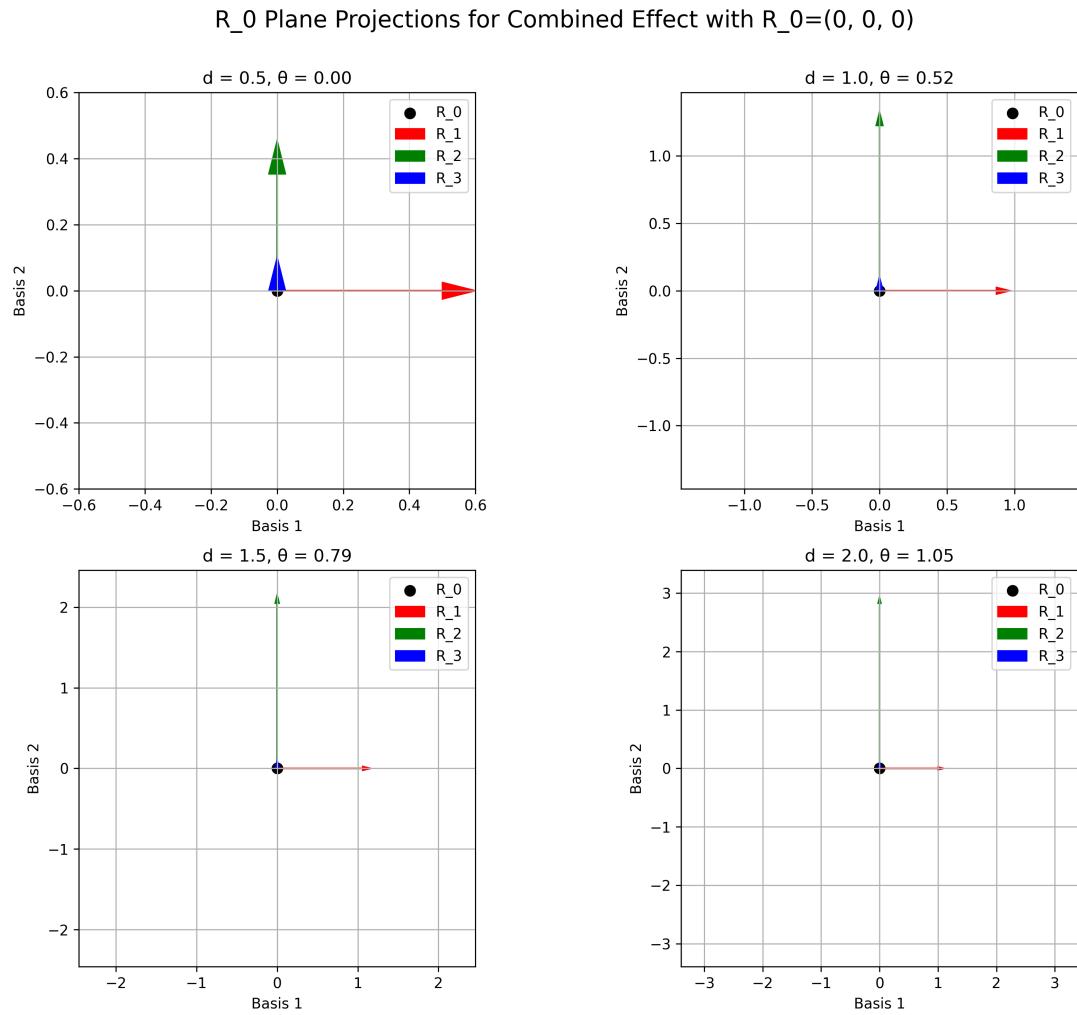


Figure 27: R_0 plane projections for combined effect with origin at $(0, 0, 0)$

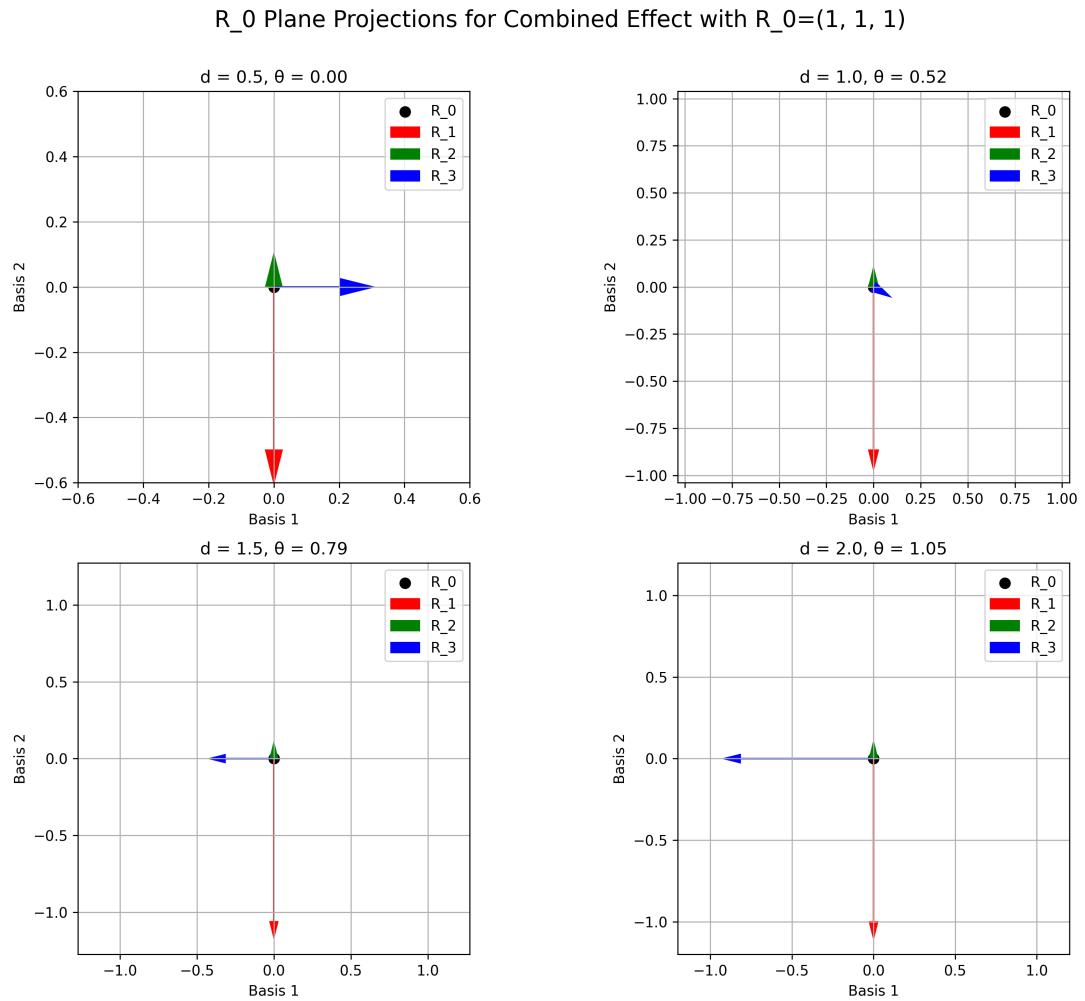


Figure 28: R_0 plane projections for combined effect with origin at $(1, 1, 1)$

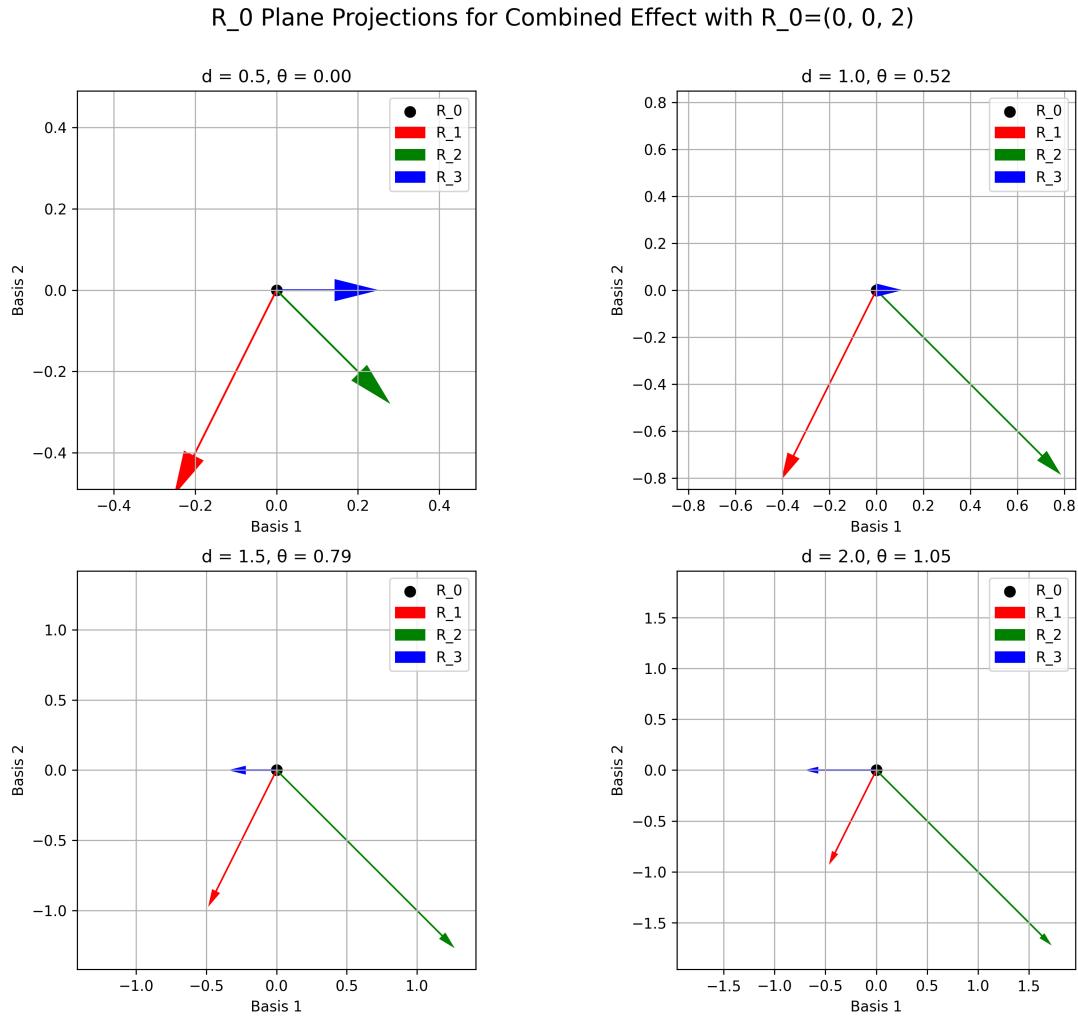


Figure 29: R_0 plane projections for combined effect with origin at $(0, 0, 2)$

9.6.3 Combined 3D and R_0 Plane Views

The combined views show both the 3D vectors and their R_0 plane projections side by side, providing a comprehensive understanding of the vector relationships.

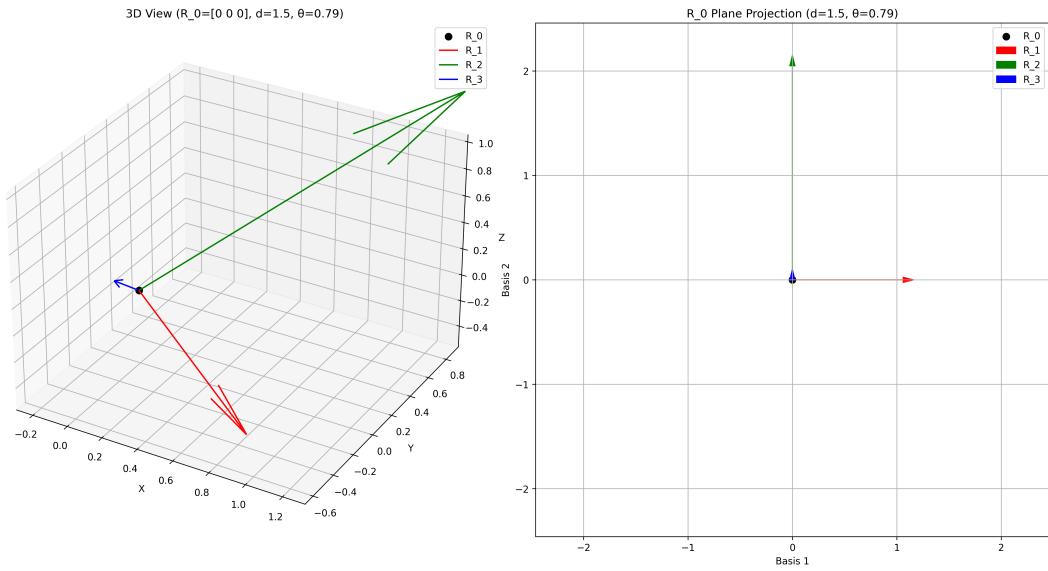


Figure 30: Combined 3D and R_0 plane view with origin at $(0,0,0)$, $d = 1.5$, $\theta = \pi/4$

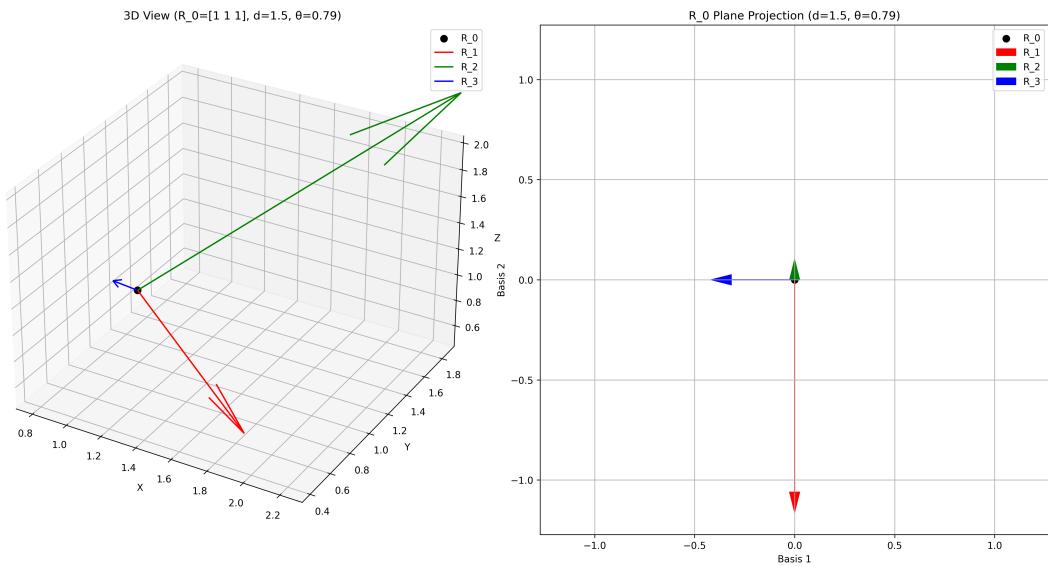


Figure 31: Combined 3D and R_0 plane view with origin at $(1,1,1)$, $d = 1.5$, $\theta = \pi/4$

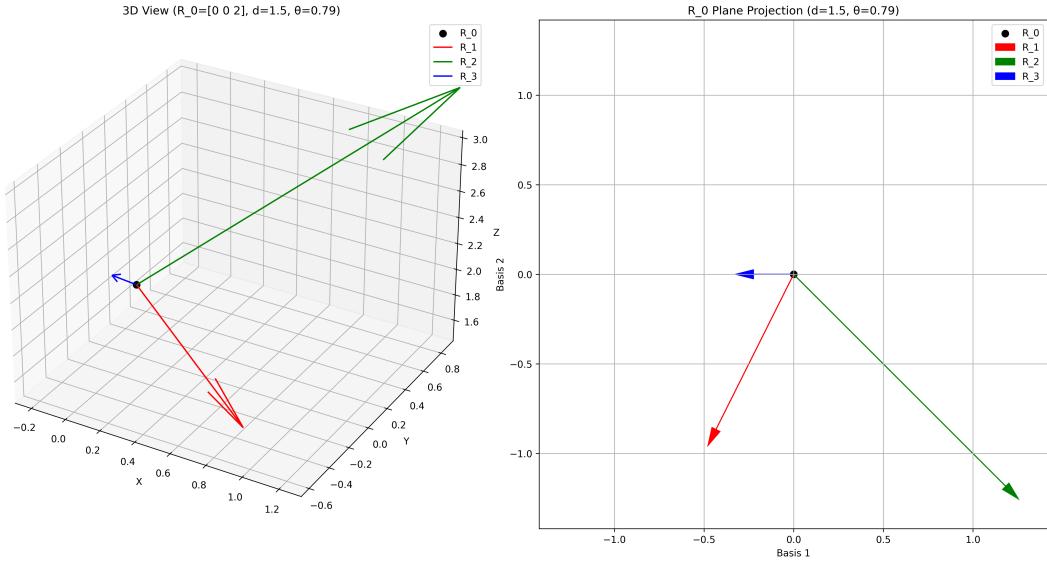


Figure 32: Combined 3D and R_0 plane view with origin at $(0, 0, 2)$, $d = 1.5$, $\theta = \pi/4$

Effect of Origin and Combined Parameters: The origin parameter \vec{R}_0 shifts the entire vector system, preserving the orthogonality of the displacement vectors. Different origin points result in different positions of the vectors in space. The figures above demonstrate how different combinations of distance and angle parameters affect the vector visualization for each origin point. This illustrates the flexibility of the generalized orthogonal vectors implementation in creating various vector configurations.

9.7 Summary of Results

The example results demonstrate that the Generalized Orthogonal Vectors Generator and Visualizer successfully generates and visualizes orthogonal vectors for various configurations. The vectors are confirmed to be orthogonal by calculating their dot products, which are all zero (within numerical precision).

The visualizations show the vectors in both 3D and 2D projections, providing different perspectives on their spatial relationships. The effects of the distance parameter, angle parameter, and origin on the vector system are also demonstrated.

10 Conclusion

The Generalized Orthogonal Vectors Generator and Visualizer package provides a comprehensive solution for generating and visualizing orthogonal vectors in three-dimensional space. This document has described the mathematical formulation, implementation details, API reference, usage examples, visualization techniques, configuration system, command-line interface, and example results of the package.

10.1 Summary of Features

The package offers the following key features:

- **Mathematical Rigor:** The package is based on a mathematically proven formulation for generating orthogonal vectors, ensuring the correctness of the results.
- **Modular Architecture:** The package is organized into separate modules for vector calculations, visualization, and configuration management, making it easy to maintain, extend, and reuse.
- **Configurability:** All aspects of vector generation and visualization can be configured through a unified configuration system, allowing for customization without modifying the code.
- **Command-line Interface:** The package provides a comprehensive command-line interface that allows users to generate and visualize orthogonal vectors without writing Python code.

- **Configuration File Support:** Configurations can be saved to and loaded from JSON files, making it easy to reuse configurations across different runs.
- **Multiple Visualization Options:** The package supports both 3D visualization and various 2D projections, providing different perspectives on the vectors.
- **Plot Saving:** Plots can be saved to files instead of being displayed interactively, allowing for the creation of visualizations for documentation or presentations.
- **Python Package:** The package can be used as a Python package, allowing for integration into other projects.

10.2 Potential Applications

The Generalized Orthogonal Vectors Generator and Visualizer package can be used in various applications, including:

- **Educational Tools:** The package can be used as an educational tool for teaching concepts related to vectors, orthogonality, and three-dimensional geometry.
- **Scientific Visualization:** The package can be used for visualizing orthogonal vectors in scientific applications, such as physics simulations or computational geometry.
- **Computer Graphics:** The package can be used in computer graphics applications that require orthogonal coordinate systems, such as camera positioning or object orientation.
- **Robotics:** The package can be used in robotics applications that require orthogonal coordinate systems, such as robot arm positioning or sensor orientation.

10.3 Future Work

The Generalized Orthogonal Vectors Generator and Visualizer package can be extended in various ways, including:

- **Additional Visualization Options:** The package could be extended to support additional visualization options, such as interactive 3D visualization or animation of vector rotation.
- **More Advanced Configuration Management:** The configuration management system could be extended to support more advanced features, such as configuration validation or configuration inheritance.
- **Integration with Other Packages:** The package could be integrated with other Python packages for scientific computing or visualization, such as SciPy or Plotly.
- **Web Interface:** The package could be extended to provide a web interface for generating and visualizing orthogonal vectors, making it accessible to users without Python knowledge.
- **Performance Optimization:** The package could be optimized for performance, especially for applications that require generating and visualizing a large number of vectors.
- **Unit Tests:** The package could be extended with comprehensive unit tests to ensure the correctness of the implementation.
- **Documentation Improvements:** The documentation could be improved with more examples, tutorials, and explanations of the mathematical concepts.

10.4 Conclusion

The Generalized Orthogonal Vectors Generator and Visualizer package provides a powerful and flexible tool for generating and visualizing orthogonal vectors in three-dimensional space. Its modular architecture, configurability, and comprehensive features make it suitable for a wide range of applications, from educational tools to scientific visualization. The package is designed to be easy to use, both as a command-line tool and as a Python package, making it accessible to users with different levels of programming experience.

A Source Code

This appendix contains the complete source code for the Generalized Orthogonal Vectors Generator and Visualizer package.

A.1 main.py

```

1 #!/usr/bin/env python3
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import argparse
5 import math
6 import os
7 import sys
8
9 from vector_utils import create_orthogonal_vectors, check_orthogonality
10 from visualization import plot_vectors_3d, plot_vectors_2d_projection,
11     plot_all_projections
12 from config import VectorConfig, default_config
13
14 def parse_arguments():
15     """
16     Parse command line arguments
17
18     Returns:
19     argparse.Namespace: Parsed arguments
20     """
21
22     parser = argparse.ArgumentParser(description='Generate and visualize orthogonal
23                                     vectors')
24
25     # Vector parameters
26     parser.add_argument('--origin', '-R', type=float, nargs=3, default=[0, 0, 0],
27                         help='Origin vector R_0 (x y z)')
28     parser.add_argument('--distance', '-d', type=float, default=1,
29                         help='Distance parameter d')
30     parser.add_argument('--angle', '-a', type=float, default=math.pi/4,
31                         help='Angle parameter theta in radians')
32
33     # Visualization parameters
34     parser.add_argument('--no-r0-plane', action='store_false', dest='show_r0_plane',
35                         help='Do not show the R_0 plane projection')
36     parser.add_argument('--no-legend', action='store_false', dest='show_legend',
37                         help='Do not show the legend')
38     parser.add_argument('--no-grid', action='store_false', dest='show_grid',
39                         help='Do not show the grid')
40
41     # Output parameters
42     parser.add_argument('--save-plots', action='store_true',
43                         help='Save plots to files instead of displaying them')
44     parser.add_argument('--output-dir', type=str, default='plots',
45                         help='Directory to save plots to')
46     parser.add_argument('--config', type=str,
47                         help='Path to configuration file')
48     parser.add_argument('--save-config', type=str,
49                         help='Save configuration to file')
50
51     return parser.parse_args()
52
53 def display_help():
54     """
55     Display detailed help information
56     """
57
58     help_text = """
59     Orthogonal Vectors Generator and Visualizer
60     =====
61
62     This tool generates and visualizes three orthogonal vectors from a given origin
63     point.
64
65     Basic Usage:
66     -----
67     python main.py
68
69     # Use default parameters

```

```

63     python main.py -R 1 1 1           # Set origin to (1,1,1)
64     python main.py -R 0 0 2 -d 1.5 -a 0.5236   # Custom origin, distance and angle
65     python main.py --help             # Show help
66
67     Parameters:
68     -----
69     -R, --origin X Y Z      : Set the origin vector R_0 coordinates (default: 0 0 0)
70     -d, --distance VALUE    : Set the distance parameter (default: 1)
71     -a, --angle VALUE       : Set the angle parameter in radians (default: \pi/4)
72
73     Visualization Options:
74     -----
75     --no-r0-plane          : Do not show the R_0 plane projection
76     --no-legend            : Do not show the legend
77     --no-grid              : Do not show the grid
78
79     Output Options:
80     -----
81     --save-plots           : Save plots to files instead of displaying them
82     --output-dir DIR       : Directory to save plots to (default: 'plots')
83
84     Configuration:
85     -----
86     --config FILE          : Load configuration from a JSON file
87     --save-config FILE     : Save current configuration to a JSON file
88
89     Examples:
90     -----
91     # Generate vectors with origin at (1,1,1), distance 2, and angle \pi/3
92     python main.py -R 1 1 1 -d 2 -a 1.047
93
94     # Save plots to a custom directory
95     python main.py -R 0 0 2 --save-plots --output-dir my_plots
96
97     # Load configuration from a file
98     python main.py --config my_config.json
99     """
100    print(help_text)
101    sys.exit(0)
102
103 def main():
104     """
105     Main function
106     """
107
108     # Check for detailed help command
109     if len(sys.argv) > 1 and sys.argv[1] == 'help':
110         display_help()
111
112     # Parse command line arguments
113     args = parse_arguments()
114
115     # Load configuration
116     if args.config:
117         config = VectorConfig.load_from_file(args.config)
118     else:
119         # Create configuration from command line arguments
120         config = VectorConfig(
121             R_0=args.origin,
122             d=args.distance,
123             theta=args.angle,
124             show_r0_plane=args.show_r0_plane,
125             show_legend=args.show_legend,
126             show_grid=args.show_grid
127         )
128
129     # Save configuration if requested
130     if args.save_config:
131         config.save_to_file(args.save_config)
132
133     # Create the orthogonal vectors
134     R_0 = config.R_0
135     R_1, R_2, R_3 = create_orthogonal_vectors(R_0, config.d, config.theta)

```

```

136     # Print vector information
137     print("R_0:", R_0)
138     print("R_1:", R_1)
139     print("R_2:", R_2)
140     print("R_3:", R_3)
141
142     # Check orthogonality
143     orthogonality = check_orthogonality(R_0, R_1, R_2, R_3)
144     print("\nChecking orthogonality (dot products should be close to zero):")
145     for key, value in orthogonality.items():
146         print(f"{key}: {value}")
147
148     # Plot the vectors
149     plots = plot_all_projections(
150         R_0, R_1, R_2, R_3,
151         show_r0_plane=config.show_r0_plane,
152         figsize_3d=config.figsize_3d,
153         figsize_2d=config.figsize_2d
154     )
155
156     # Save or show the plots
157     if args.save_plots:
158         # Create output directory if it doesn't exist
159         os.makedirs(args.output_dir, exist_ok=True)
160
161         # Save each plot
162         for name, (fig, _) in plots.items():
163             filename = os.path.join(args.output_dir, f"{name}.png")
164             fig.savefig(filename)
165             print(f"Saved plot to {filename}")
166     else:
167         # Show the plots
168         plt.show()
169
170 if __name__ == "__main__":
171     main()

```

A.2 vector_utils.py

```

1 #!/usr/bin/env python3
2 import numpy as np
3
4 def create_orthogonal_vectors(R_0=(0, 0, 0), d=1, theta=0):
5     """
6         Create 3 orthogonal R vectors for R_0
7
8     Parameters:
9         R_0 (tuple or numpy.ndarray): The origin vector, default is (0, 0, 0)
10        d (float): The distance parameter, default is 1
11        theta (float): The angle parameter in radians, default is 0
12
13    Returns:
14        tuple: Three orthogonal vectors R_1, R_2, R_3
15    """
16
17    # Convert R_0 to numpy array for vector operations
18    R_0 = np.array(R_0)
19
20    # Calculate R_1, R_2, R_3 according to the given formulas
21    # R_1 = R_0 + d * (cos(theta))*sqrt(2/3)
22    R_1 = R_0 + d * np.cos(theta) * np.sqrt(2/3) * np.array([1, -1/2, -1/2])
23
24    # R_2 = R_0 + d * (cos(theta)/sqrt(3) + sin(theta))/sqrt(2)
25    R_2 = R_0 + d * (np.cos(theta)/np.sqrt(3) + np.sin(theta))/np.sqrt(2) * np.array([1, 1, 1])
26
27    # R_3 = R_0 + d * (sin(theta) - cos(theta)/sqrt(3))/sqrt(2)
28    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)
29
30    return R_1, R_2, R_3
31
32 def check_orthogonality(R_0, R_1, R_2, R_3):

```

```

32     """
33     Check if the vectors R_1, R_2, R_3 are orthogonal with respect to R_0
34
35     Parameters:
36     R_0, R_1, R_2, R_3 (numpy.ndarray): The vectors to check
37
38     Returns:
39     dict: Dictionary containing the dot products between pairs of vectors
40     """
41     dot_1_2 = np.dot(R_1 - R_0, R_2 - R_0)
42     dot_1_3 = np.dot(R_1 - R_0, R_3 - R_0)
43     dot_2_3 = np.dot(R_2 - R_0, R_3 - R_0)
44
45     return {
46         "R_1 $\cdot$ R_2": dot_1_2,
47         "R_1 $\cdot$ R_3": dot_1_3,
48         "R_2 $\cdot$ R_3": dot_2_3
49     }

```

A.3 config.py

```

1 #!/usr/bin/env python3
2 import numpy as np
3 import math
4 import json
5 import os
6
7 class VectorConfig:
8     """
9         Configuration class for orthogonal vector generation and visualization
10    """
11    def __init__(self,
12                 R_0=(0, 0, 0),
13                 d=1,
14                 theta=math.pi/4,
15                 show_r0_plane=True,
16                 figsize_3d=(10, 8),
17                 figsize_2d=(8, 8),
18                 show_legend=True,
19                 show_grid=True):
20        """
21            Initialize the configuration
22
23        Parameters:
24        R_0 (tuple or list): The origin vector
25        d (float): The distance parameter
26        theta (float): The angle parameter in radians
27        show_r0_plane (bool): Whether to show the R_0 plane projection
28        figsize_3d (tuple): Figure size for 3D plot
29        figsize_2d (tuple): Figure size for 2D plots
30        show_legend (bool): Whether to show the legend
31        show_grid (bool): Whether to show the grid
32        """
33        self.R_0 = np.array(R_0)
34        self.d = d
35        self.theta = theta
36        self.show_r0_plane = show_r0_plane
37        self.figsize_3d = figsize_3d
38        self.figsize_2d = figsize_2d
39        self.show_legend = show_legend
40        self.show_grid = show_grid
41
42    def to_dict(self):
43        """
44            Convert the configuration to a dictionary
45
46        Returns:
47        dict: Dictionary representation of the configuration
48        """
49        return {
50            'R_0': self.R_0.tolist(),
51            'd': self.d,

```

```

52         'theta': self.theta,
53         'show_r0_plane': self.show_r0_plane,
54         'figsize_3d': self.figsize_3d,
55         'figsize_2d': self.figsize_2d,
56         'show_legend': self.show_legend,
57         'show_grid': self.show_grid
58     }
59
60     @classmethod
61     def from_dict(cls, config_dict):
62         """
63             Create a configuration from a dictionary
64
65             Parameters:
66             config_dict (dict): Dictionary containing configuration parameters
67
68             Returns:
69             VectorConfig: Configuration object
70             """
71     return cls(
72         R_0=config_dict.get('R_0', (0, 0, 0)),
73         d=config_dict.get('d', 1),
74         theta=config_dict.get('theta', math.pi/4),
75         show_r0_plane=config_dict.get('show_r0_plane', True),
76         figsize_3d=config_dict.get('figsize_3d', (10, 8)),
77         figsize_2d=config_dict.get('figsize_2d', (8, 8)),
78         show_legend=config_dict.get('show_legend', True),
79         show_grid=config_dict.get('show_grid', True)
80     )
81
82     def save_to_file(self, filename):
83         """
84             Save the configuration to a JSON file
85
86             Parameters:
87             filename (str): Path to the output file
88             """
89         with open(filename, 'w') as f:
90             json.dump(self.to_dict(), f, indent=4)
91
92     @classmethod
93     def load_from_file(cls, filename):
94         """
95             Load a configuration from a JSON file
96
97             Parameters:
98             filename (str): Path to the input file
99
100            Returns:
101            VectorConfig: Configuration object
102            """
103        if not os.path.exists(filename):
104            print(f"Warning: Config file {filename} not found. Using default
105 configuration.")
106            return cls()
107
108        with open(filename, 'r') as f:
109            config_dict = json.load(f)
110
111        return cls.from_dict(config_dict)
112
113 # Default configuration
114 default_config = VectorConfig()

```

A.4 visualization.py

```

1  #!/usr/bin/env python3
2  import numpy as np
3  import matplotlib.pyplot as plt
4  from mpl_toolkits.mplot3d import Axes3D
5
6  def plot_vectors_3d(R_0, R_1, R_2, R_3, figsize=(10, 8), show_legend=True):

```

```

7     """
8     Plot the vectors in 3D
9
10    Parameters:
11        R_0 (numpy.ndarray): The origin vector
12        R_1, R_2, R_3 (numpy.ndarray): The three orthogonal vectors
13        figsize (tuple): Figure size (width, height) in inches
14        show_legend (bool): Whether to show the legend
15
16    Returns:
17        tuple: (fig, ax) matplotlib figure and axis objects
18    """
19    fig = plt.figure(figsize=figsize)
20    ax = fig.add_subplot(111, projection='3d')
21
22    # Plot the origin
23    ax.scatter(R_0[0], R_0[1], R_0[2], color='black', s=100, label='R_0')
24
25    # Plot the vectors as arrows from the origin
26    vectors = [R_1, R_2, R_3]
27    colors = ['r', 'g', 'b']
28    labels = ['R_1', 'R_2', 'R_3']
29
30    for i, (vector, color, label) in enumerate(zip(vectors, colors, labels)):
31        ax.quiver(R_0[0], R_0[1], R_0[2],
32                   vector[0]-R_0[0], vector[1]-R_0[1], vector[2]-R_0[2],
33                   color=color, label=label, arrow_length_ratio=0.1)
34
35    # Set labels and title
36    ax.set_xlabel('X')
37    ax.set_ylabel('Y')
38    ax.set_zlabel('Z')
39    ax.set_title('3D Plot of Orthogonal Vectors')
40
41    # Set equal aspect ratio
42    max_range = np.array([
43        np.max([R_0[0], R_1[0], R_2[0], R_3[0]]) - np.min([R_0[0], R_1[0], R_2[0], R_3[0]]),
44        np.max([R_0[1], R_1[1], R_2[1], R_3[1]]) - np.min([R_0[1], R_1[1], R_2[1], R_3[1]]),
45        np.max([R_0[2], R_1[2], R_2[2], R_3[2]]) - np.min([R_0[2], R_1[2], R_2[2], R_3[2]])
46    ]).max() / 2.0
47
48    mid_x = (np.max([R_0[0], R_1[0], R_2[0], R_3[0]]) + np.min([R_0[0], R_1[0], R_2[0], R_3[0]])) / 2
49    mid_y = (np.max([R_0[1], R_1[1], R_2[1], R_3[1]]) + np.min([R_0[1], R_1[1], R_2[1], R_3[1]])) / 2
50    mid_z = (np.max([R_0[2], R_1[2], R_2[2], R_3[2]]) + np.min([R_0[2], R_1[2], R_2[2], R_3[2]])) / 2
51
52    ax.set_xlim(mid_x - max_range, mid_x + max_range)
53    ax.set_ylim(mid_y - max_range, mid_y + max_range)
54    ax.set_zlim(mid_z - max_range, mid_z + max_range)
55
56    if show_legend:
57        ax.legend()
58
59    return fig, ax
60
61 # Note: This is a partial listing. The full visualization.py file contains additional
62 # functions
63 # such as plot_vectors_2d_projection and plot_all_projections that are omitted here for
64 # brevity.

```

A.5 __init__.py

```

1 # Generalized Orthogonal Vectors Generator and Visualizer
2 # This package provides tools for generating and visualizing orthogonal vectors
3
4 from .vector_utils import create_orthogonal_vectors, check_orthogonality

```

```
5 from .visualization import plot_vectors_3d, plot_vectors_2d_projection,
6     plot_all_projections
7 from .config import VectorConfig, default_config
8
9 __all__ = [
10     'create_orthogonal_vectors',
11     'check_orthogonality',
12     'plot_vectors_3d',
13     'plot_vectors_2d_projection',
14     'plot_all_projections',
15     'VectorConfig',
16     'default_config'
17 ]
18 __version__ = '1.0.0'
```