Change of Basis and Coordinates of vectors

Meenu S

January 5, 2026

1 Introduction

This document provides a detailed explanation of a Python program that computes the change of basis between two vector spaces, calculates the transition matrix, and visualizes the result. The program performs the following tasks:

- Accepts the vector space dimension (2D or 3D).
- Prompts the user for two basis vectors.
- Computes the transition matrix from the initial basis to the new basis.
- Visualizes the vector in both bases, and generates plots.

2 Python Code Listing

```
import numpy as np
import matplotlib.pyplot as plt
import os

def get_basis_from_user(dim, basis_name):
    """Prompts the user to enter basis vectors."""
    print(f"--- Enter the {dim} {basis_name} vectors ---")
    basis_vectors = []
    for i in range(dim):
        while True:
            try:
                b_str = input(f"Basis vector {i+1}: ").split()
                if len(b_str) != dim:
                    print(f"Error: Please enter {dim} components.")
                    continue
                basis_vectors.append([float(x) for x in b_str])
                break
            except ValueError:
                print("Invalid input. Please enter numbers only.")
```

```
    # Create matrix with basis vectors as columns
    matrix = np.array(basis_vectors).T
    # Check if the basis is valid (i.e., vectors are linearly independent)
    if abs(np.linalg.det(matrix)) < 1e-9: # Use a tolerance for floating point numbers
        print(f"\nError: The vectors provided for '{basis_name}' are not linearly independent and do not form a valid basis.")
        return None
    return matrix

def general_change_of_basis():
    """
    Performs a change of basis for a vector from one arbitrary basis to another.
    """
    try:
        # 1. Specify the dimension
        dim_str = input("Enter the dimension of the vector space (2 or 3): ")
        dim = int(dim_str)
        if dim not in [2, 3]:
            print("Invalid input. This script can only plot for dimensions 2 and 3.")
            return

        # 2. Get the initial basis (B)
        B = get_basis_from_user(dim, "initial basis (B)")
        if B is None: return

        # 3. Get the vector's coordinates in the initial basis
        print("\nEnter the vector's coordinates relative to the initial basis B, separated by spaces:")
        v_coords_B_str = input("Vector coordinates: ").split()
        v_coords_B = np.array([float(x) for x in v_coords_B_str]).reshape(-1, 1) # Column vector

        # 4. Get the new basis (B')
        B_prime = get_basis_from_user(dim, "new basis (B')")
        if B_prime is None: return

        # 5. Define the transition matrix C = (B')^-1 * B
        B_prime_inv = np.linalg.inv(B_prime)
        C = B_prime_inv @ B

```

```
        # 6. Calculate the new coordinates: v_coords_B' = C * v_coords_B
        v_coords_B_prime = C @ v_coords_B

        # 7. Print the results
        np.set_printoptions(precision=3, suppress=True)
        print("\n" + "="*40)
        print("               RESULTS")
        print("="*40)
        print(f"Transition Matrix from B to B' (C):\n{C}\n")
        print(f"Original coordinates in basis B: {v_coords_B.flatten()}")
        print(f"New coordinates in basis B': {v_coords_B_prime.flatten()}")
        print("="*40)

        # 8. Generate plots for 2D or 3D cases
        # The actual geometric vector in standard coordinates is B @ v_coords_B
        v_geometric = (B @ v_coords_B).flatten()

        print(f"\nGenerating {dim}D plots...")
        fig = plt.figure(figsize=(16, 8))
        save_path = 'general_basis_change.png'

        if dim == 2:
            ax1 = fig.add_subplot(1, 2, 1)
            ax2 = fig.add_subplot(1, 2, 2)

            # --- Plot 1: Vector in Initial Basis B ---
            ax1.set_title("Vector in Initial Basis (B)")
            ax1.quiver(0, 0, B[0, 0], B[1, 0], angles='xy', scale_units='xy', scale=1, color='g', label=f'b1 = {B[:,0]}')
            ax1.quiver(0, 0, B[0, 1], B[1, 1], angles='xy', scale_units='xy', scale=1, color='b', label=f'b2 = {B[:,1]}')
            ax1.quiver(0, 0, v_geometric[0], v_geometric[1], angles='xy', scale_units='xy', scale=1, color='r', label=f'Coords = {v_coords_B.flatten()}')

            # --- Plot 2: Vector in New Basis B' ---
            ax2.set_title("Vector in New Basis (B')")
            ax2.quiver(0, 0, B_prime[0, 0], B_prime[1, 0], angles='xy', scale_units='xy', scale=1, color='g', label=f"b'1 = {B_prime[:,0]}")
```

```
            ax2.quiver(0, 0, B_prime[0, 1], B_prime[1, 1], angles='xy', scale_units='xy', scale=1, color='b', label=f"b'2 = {B_prime[:,1]}")
            ax2.quiver(0, 0, v_geometric[0], v_geometric[1], angles='xy', scale_units='xy', scale=1, color='r', label=f"Coords = {v_coords_B_prime.flatten()}")

            all_vectors = np.hstack([B, B_prime, v_geometric.reshape(-1,1)])
            max_val = np.max(np.abs(all_vectors)) + 1
            for ax in [ax1, ax2]:
                ax.set_aspect('equal', adjustable='box')
                ax.set_xlim(-max_val, max_val); ax.set_ylim(-max_val, max_val)
                ax.grid(); ax.legend()
                ax.axhline(0, color='k', lw=0.5); ax.axvline(0, color='k', lw=0.5)

        elif dim == 3:
            ax1 = fig.add_subplot(1, 2, 1, projection='3d')
            ax2 = fig.add_subplot(1, 2, 2, projection='3d')

            # --- Plot 1: Vector in Initial Basis B ---
            ax1.set_title("Vector in Initial Basis (B)")
            ax1.quiver(0, 0, 0, B[0,0], B[1,0], B[2,0], color='g', label=f'b1')
            ax1.quiver(0, 0, 0, B[0,1], B[1,1], B[2,1], color='b', label=f'b2')
            ax1.quiver(0, 0, 0, B[0,2], B[1,2], B[2,2], color='y', label=f'b3')
            ax1.quiver(0, 0, 0, v_geometric[0], v_geometric[1], v_geometric[2], color='r', label=f'Coords = {v_coords_B.flatten()}')

            # --- Plot 2: Vector in New Basis B' ---
            ax2.set_title("Vector in New Basis (B')")
            ax2.quiver(0, 0, 0, B_prime[0,0], B_prime[1,0], B_prime[2,0], color='g', label=f"b'1")
            ax2.quiver(0, 0, 0, B_prime[0,1], B_prime[1,1], B_prime[2,1], color='b', label=f"b'2")
            ax2.quiver(0, 0, 0, B_prime[0,2], B_prime[1,2], B_prime[2,2], color='y', label=f"b'3")
            ax2.quiver(0, 0, 0, v_geometric[0], v_geometric[1], v_geometric[2], color='r', label=f"Coords = {v_coords_B_prime.flatten()}")

```

```
            all_vectors = np.hstack([B, B_prime, v_geometric.reshape(-1,1)])
            max_val = np.max(np.abs(all_vectors)) + 1
            for ax in [ax1, ax2]:
                ax.set_xlim(-max_val, max_val); ax.set_ylim(-max_val, max_val); ax.set_zlim(-max_val, max_val)
                ax.set_xlabel('X'); ax.set_ylabel('Y'); ax.set_zlabel('Z')
                ax.legend()

        fig.tight_layout(pad=3.0)
        try:
            plt.savefig(save_path)
            if os.path.exists(save_path):
                print(f"Success! Plot saved as {save_path}.")
            else:
                print(f"Error: Plot file not created.")
        except Exception as e:
            print(f"--- FAILED TO SAVE PLOT: {e} ---")

    except ValueError:
        print("Invalid input. Please ensure you enter numbers only.")
    except np.linalg.LinAlgError as e:
        print(f"A matrix error occurred: {e}. One of the bases might be invalid.")
    except Exception as e:
        import traceback
        print(f"--- AN UNEXPECTED ERROR OCCURRED: {e} ---")
        print(f"TRACEBACK: {traceback.format_exc()}")

if __name__ == '__main__':
    # Activate the virtual environment first by running:
    # source .venv/bin/activate
    general_change_of_basis()
```

Listing 1: Python Program for Change of Basis and Transition Matrices

OUTPUT

Enter the dimension of the vector space (2 or 3): 3
— Enter the 3 initial basis (B) vectors —
Basis vector 1: 1 1 1
Basis vector 2: 1 -1 1
Basis vector 3: 0 0 1

Enter the vector's coordinates relative to the initial basis B, separated by spaces:
Vector coordinates: 6 -1 -1

— Enter the 3 new basis (B') vectors —
Basis vector 1: 2 2 0
Basis vector 2: 0 1 1
Basis vector 3: 1 0 1

Transition Matrix from B to B' (C):

$$\begin{bmatrix} 0.25 & -0.25 & -0.25 \\ 0.5 & -0.5 & 0.5 \\ 0.5 & 1.5 & 0.5 \end{bmatrix}$$

Original coordinates in basis B: [6. -1. -1.]
New coordinates in basis B': [2. 3. 1.]

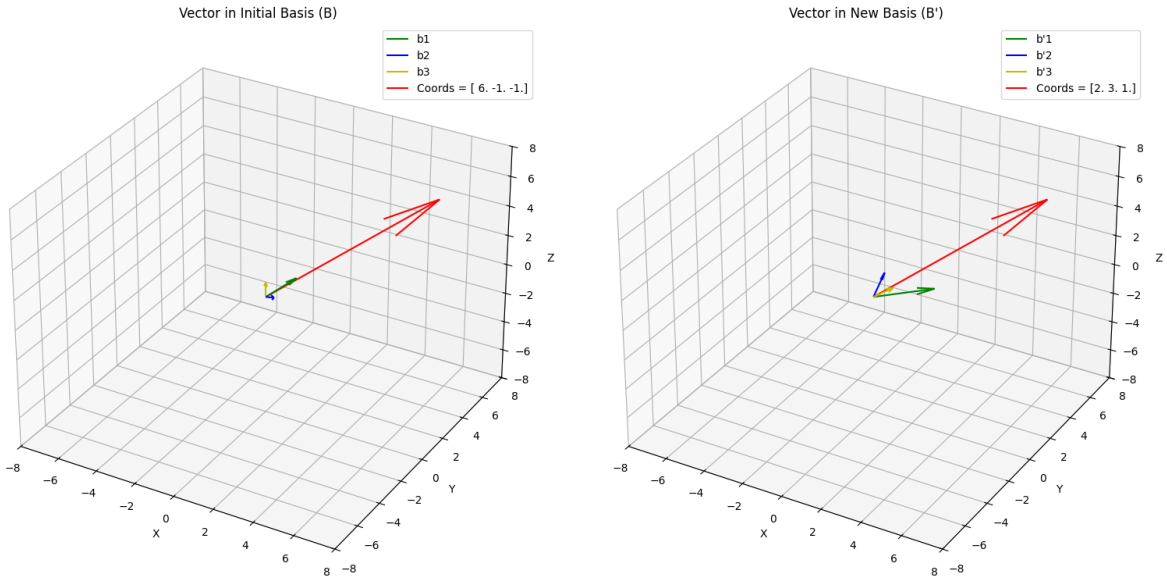

Figure 1: Representation of the vector in 2 different bases

3 Explanation of the Code (Notes)

Import Statements

- `import numpy as np`: Imports the NumPy package for numerical operations, especially matrix and vector calculations.
- `import matplotlib.pyplot as plt`: Imports Matplotlib for generating plots of vectors in the different bases.

- `import os`: Used to check if the plot file is successfully saved.

The Function `get_basis_from_user`

- This function prompts the user to enter basis vectors.
- The matrix is created using these vectors as columns. It checks if the basis vectors are linearly independent by checking if the determinant of the matrix is non-zero.
- If the determinant is zero, the vectors are linearly dependent, and the function returns `None`.

The Function `general_change_of_basis`

- This function handles the entire process of changing the basis, including obtaining the initial and new bases, calculating the transition matrix, and visualizing the result.

Step-by-step Breakdown

- `dim = int(input("Enter the dimension of the vector space (2 or 3): "))`: Takes the dimension of the vector space as input (either 2D or 3D).
- `B = get_basis_from_user(dim, "initial basis (B)")`: Calls the `get_basis_from_user` function to obtain the initial basis vectors.
- $\mathtt{v}_c oords_B = np.array([float(x) for x in v\_coords\_B\_str]).reshape(-1, 1) : Convertstheinputvectorcoo$
- `C = np.linalg.inv(B_prime) @ B`: Calculates the transition matrix C, where $C = (B')^{-1} \times B$.
- `v_coords_B_prime = C @ v_coords_B`: Calculates the vector's new coordinates in the new basis by multiplying the transition matrix with the original coordinates
- The program then prints the transition matrix and the original and new coordinates.
- Finally, it generates 2D or 3D plots of the vectors in the initial and new bases using Matplotlib.