Exploring Linear Algebra in Python with NumPy - Beginner's Series Part 3

Welcome to Part 3 of our NumPy series. In the first two parts, we explored various foundational aspects of NumPy.

- Part 1, covered the creation of NumPy arrays, arithmetic operations, and the crucial role of broadcasting in facilitating arithmetic operations for arrays of different shapes.
- Part 2 , delved into statistical operations, scientific operations, trigonometric operations, array manipulation, and slicing.

In this installment, we will focus on the application of NumPy in Linear Algebra, an area integral to data science, machine learning, and many fields of engineering.

Introduction to Scalars, Vectors, and Matrices

Scalars : A scalar is a single numerical value, typically represented in mathematics and engineering by a lowercase letter (e.g., a,b,c). In the context of linear algebra, scalars are often used to define magnitude or size, and they can scale vectors and matrices by multiplication

A scalar example can be a simple number like 5 or 3.14, used to scale a vector or matrix in operations.

Vectors: A vector represents a quantity that has both magnitude and direction. In the context of linear algebra, we often work with column vectors, which are represented as one-dimensional arrays in NumPy.

In [1]:
```python
import numpy as np
# Example of a column vector
vector = np.array([[1], [2], [3]])
print("Column Vector:\n", vector)

# Example of a row vector
vector = np.array([1, 2, 3])
print("Row Vector:\n", vector)
```

```
Column Vector:
 [[1]
 [2]
 [3]]
Row Vector:
 [1 2 3]
```

Matrices : A matrix is a rectangular array of numbers arranged in rows and columns. It generalizes the concept of vectors to multiple dimensions. Matrices are used in linear algebra to perform operations involving linear transformations, systems of linear equations, and representation of data sets. It can be used to encode data for machine learning models, transform shapes in computer graphics, or represent complex systems in engineering. Matrices are 2D arrays in NumPy, used to represent complex linear transformations.

In [2]:
```python
import numpy as np
# Example of a matrix
matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print("Matrix:\n", matrix)
```

```
Matrix:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
```

Vector Operations:

Addition and Subtraction: We can add or subtract vectors and matrices element-wise. For example:

In [19]:
```python
# Define two vectors
vector1 = np.array([1, 2, 3])
vector2 = np.array([4, 5, 6])

# Add the vectors
result_addition = vector1 + vector2

# Subtract the vectors
result_subtraction = vector1 - vector2

print("\nVector 1:")
print(vector1)
print("\nVector 2:")
print(vector2)
print("\nVector Addition:")
print(result_addition)

print("\nVector Subtraction:")
print(result_subtraction)
```

```
Vector 1:
[1 2 3]

Vector 2:
[4 5 6]

Vector Addition:
[5 7 9]

Vector Subtraction:
[-3 -3 -3]
```

Multiplication of a Vector with a scalar:

In [21]:
```python
# Scalar multiplication
scalar = 2
result_scalar_multiply = vector1 * scalar
```

```
print ('Scalar:',scalar)
print('Vector:',vector1)
print("\nScalar Multiplication:")
print(result_scalar_multiply)
```

```
Scalar: 2
Vector: [1 2 3]

Scalar Multiplication:
[2 4 6]
```

Dot Product of two Vectors

It calculates the sum of the element-wise products of two vectors:

In [23]:
```
# Calculate the dot product of two vectors
dot_product = np.dot(vector1, vector2)
print("\nVector 1:")
print(vector1)
print("\nVector 2:")
print(vector2)
print("\nDot Product:")
print(dot_product)
```

```
Vector 1:
[1 2 3]

Vector 2:
[4 5 6]

Dot Product:
32
```

The Angle Between Two Vectors

To find the angle between two vectors in Python, you can use the dot product and trigonometric functions. The angle θ between two vectors v1 and v2 can be calculated using the following formula:

θ = arccos((v1 · v2) / (||v1|| * ||v2||))

Note: *arccosine means inverse cosine*

Where:

- v1 · v2 is the dot product of the two vectors.
- ||v1|| and ||v2|| are the magnitudes (norms) of the vectors. The norm of a vector, specifically the Euclidean norm (or L2 norm), is the length of the vector from the origin to its point in space

Let's calculate

In [52]:
```
# Define the vectors
v1 = np.array([1, 0])
v2 = np.array([1, 1])
```

```
# Calculate the dot product of v1 and v2
dot_product = np.dot(v1, v2)

# Calculate the magnitudes (norms) of v1 and v2
magnitude_v1 = np.linalg.norm(v1)
magnitude_v2 = np.linalg.norm(v2)

# Calculate the angle between the vectors in radians
angle_radians = np.arccos(dot_product / (magnitude_v1 * magnitude_v2))

# Convert the angle from radians to degrees
angle_degrees = np.degrees(angle_radians)

print('Angle between the two vectors',angle_degrees)
```

```
Angle between the two vectors 45.00000000000001
```

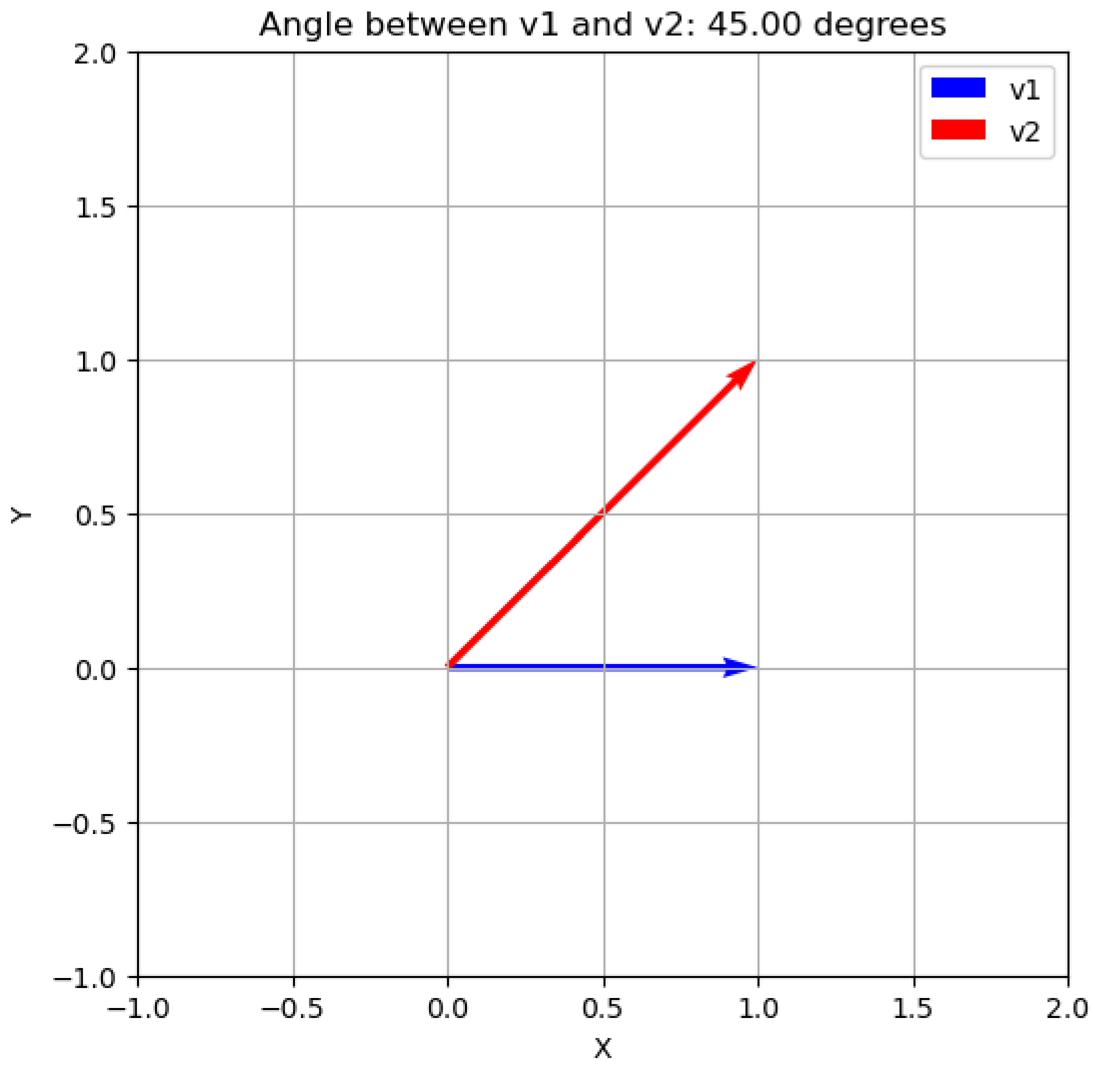
Let's Visualize these vectors and see if the result is appropriate

In [53]:

```
# Plot the vectors
import matplotlib.pyplot as plt
plt.figure(figsize=(6, 6))
plt.quiver(0, 0, v1[0], v1[1], angles='xy', scale_units='xy', scale=1, color='blue', l
plt.quiver(0, 0, v2[0], v2[1], angles='xy', scale_units='xy', scale=1, color='red', la
plt.xlim(-1, 2)
plt.ylim(-1, 2)
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()

# Display the angle between the vectors
plt.title(f'Angle between v1 and v2: {angle_degrees:.2f} degrees')

plt.grid(True)
plt.show()
```



Visually it is evident that it is 45 degree

Basic Matrix Operations

Matrix Multiplication: It involves multiplying rows of the first matrix by columns of the second matrix. NumPy provides the np.matmul for matrix multiplication:

Mathematically it is calculated as,

$$A = \begin{bmatrix} 3 & 7 \\ 4 & 9 \end{bmatrix} \ and \ B = \begin{bmatrix} 6 & 2 \\ 5 & 8 \end{bmatrix}$$

Now each of the elements of product matrix AB can be calculated as follows:

- AB_{11} = 3 × 6 + 7 ×5 = 53
- AB_{12} = 3 × 2 + 7 × 8 = 62
- AB_{21} = 4 × 6 + 9 × 5 = 69
- AB_{22} = 4 × 2 + 9 × 8 = 80

Therefore,

$$AB = \begin{bmatrix} 53 & 62 \\ 69 & 80 \end{bmatrix}$$

In [38]:
```
A = np.array([[3, 7], [4, 9]])
B = np.array([[6, 2], [5, 8]])

result = np.matmul(A, B)
print('MatrixA:\n',A)
print('MatrixA:\n',B)
print('Matrix Multiplication Result:\n',result)
```

```
MatrixA:
 [[3 7]
 [4 9]]
MatrixA:
 [[6 2]
 [5 8]]
Matrix Multiplication Result:
 [[53 62]
 [69 80]]
```

Matrix Transposition:

Transposing a matrix means switching its rows and columns. NumPy provides the T attribute for this purpose:

In [39]:
```
# Transpose a matrix
matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
transpose_matrix = matrix.T
print("\nMatrix:")
print(matrix)
print("\nMatrix Transposition:")
print(transpose_matrix)
```

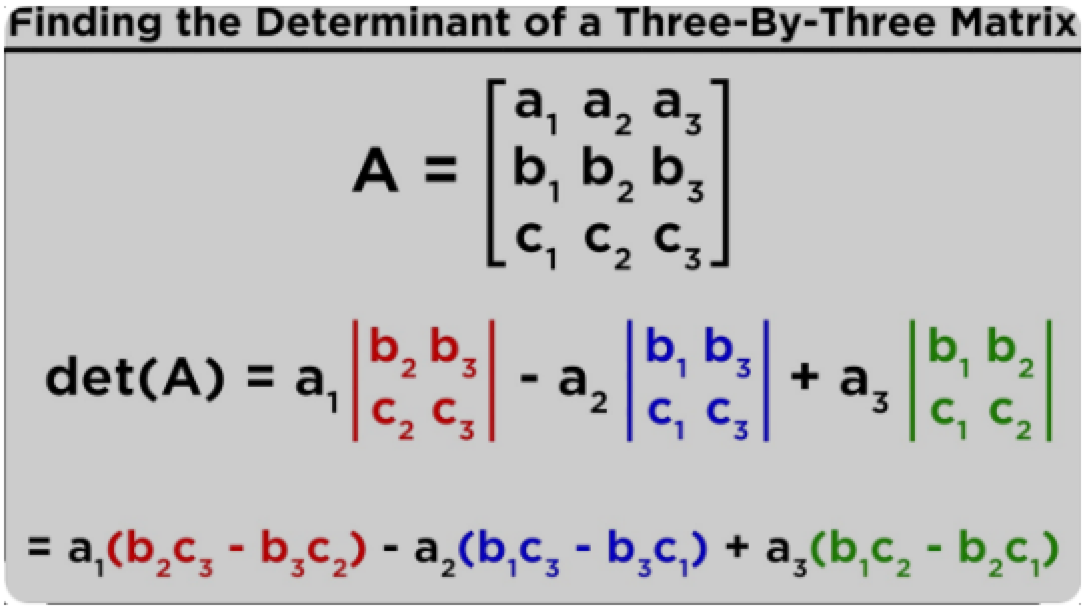
```
Matrix:
[[1 2 3]
 [4 5 6]
 [7 8 9]]

Matrix Transposition:
[[1 4 7]
 [2 5 8]
 [3 6 9]]
```

Matrix Determinant:

The determinant of a matrix is the scalar value computed for a given square matrix.It can be considered as the scaling factor for the transformation of a matrix.

Mathematically it is calculated as,



Finding the Determinant of a Three-By-Three Matrix

$$A = \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{bmatrix}$$

$$\det(A) = a_1 \begin{vmatrix} b_2 & b_3 \\ c_2 & c_3 \end{vmatrix} - a_2 \begin{vmatrix} b_1 & b_3 \\ c_1 & c_3 \end{vmatrix} + a_3 \begin{vmatrix} b_1 & b_2 \\ c_1 & c_2 \end{vmatrix}$$

$$= a_1(b_2c_3 - b_3c_2) - a_2(b_1c_3 - b_3c_1) + a_3(b_1c_2 - b_2c_1)$$

NumPy provides a convenient function, np.linalg.det(), to calculate the determinant of a square matrix. Below is how we can use it in Python,

In [51]:
```python
# Define a 2x2 matrix
matrix = np.array([[2, 4], [1, 3]])

# Calculate the determinant
det = np.linalg.det(matrix)

print("Matrix:\n",matrix)
print(f"Determinant: {det}")
```

```
Matrix:
 [[2 4]
 [1 3]]
Determinant: 2.0
```

In [50]:
```python
A = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]])

# Calculate the determinant
```

```python
det_A = np.linalg.det(A)

print("Matrix:\n",A)
print("Determinant of the matrix A:", det_A.round())
```

```
Matrix:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
Determinant of the matrix A: -0.0
```

Conclusion:

In this notebook, we've explored fundamental concepts of linear algebra using NumPy, covering scalars, vectors, matrices, and basic operations like vector addition, subtraction, multiplication by scalars, dot products. Additionally, we delved into basic matrix operations, demonstrating matrix multiplication, transposition, and determinant calculations These operations are crucial for various applications in data science, engineering, and physics.

Due to the extensive scope of linear algebra, I've decided to defer the discussion of vector transformations, eigenvalues, and eigenvectors to the next installment. This approach ensures that each topic is given the attention it deserves without overwhelming the content of a single notebook.

Stay tuned for the next part of our series, where we will dive deeper into linear transformations and explore eigenvalues and eigenvectors, enhancing your skills in handling more complex algebraic computations in data analysis and beyond.