Vector and Matrix Visualizer

This notebook provides an interactive visualization of fundamental concepts on **Linear Algebra**, including vectors, matrices, plotting and solution of different operations on vectors and matrices on 2D plane.

We do not do operations on 3D vectors here and work on 2D only.

it aims to build an intuitive geometric understanding of 2D vectors.

Mathematical Background:

A vector in two dimensions can be written as:
$v = [x, y]$
We will define vectors as row matrices throughout the notebook.

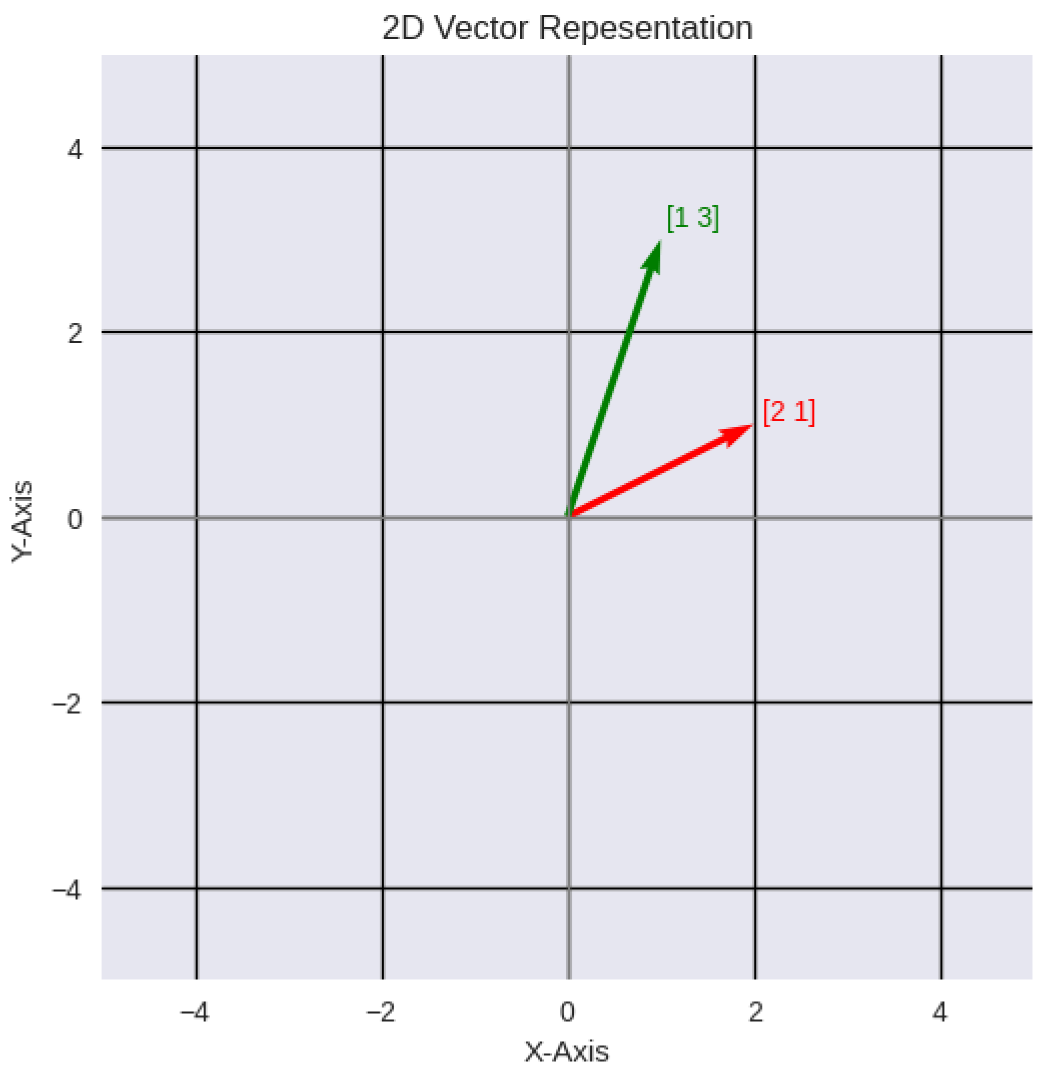
A linear transformation on space $R^2 * R^2$ can be represented by a matrix A.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('seaborn-v0_8')
```

Visualizing 2D vectors

We can represent vectors graphically as arrows starting from origin or a point (x_1, y_1) and ending at the point (x, y).

In [2]:
```
# Function for plotting vectors. This function will be used throughout th
def plot_vectors(vectors, colors=None, title="2D Vector Repesentation"):
    plt.figure(figsize=(6,6))
    plt.axhline(0, color='grey', lw=1)
    plt.axvline(0, color='grey', lw=1)
    plt.grid(True, linestyle='-', color='black')

    for i, v in enumerate(vectors):
        color = colors[i] if colors else 'blue'
        plt.quiver(0, 0, v[0], v[1], angles='xy', scale_units='xy', scale
        plt.text(v[0]*1.05, v[1]*1.05, f"{v}", color=color, fontsize=10)

    plt.xlim(-5,5)
    plt.ylim(-5,5)
    plt.xlabel("X-Axis")
    plt.ylabel("Y-Axis")
    plt.title(title)
    plt.show()

v1=np.array([2, 1]) # vector 1
v2=np.array([1, 3]) # vector 2
plot_vectors([v1, v2], colors=['r', 'g'])
```



Vector Operations:

Now, we will compute and visualize basic vector operations like addition, subtraction, scalar multiplication, dot product, line between vectors and vector projection.

- addition: In addition of two vectors we compute the resultant vector by adding two vectors:
  $\text{let } \ v1 = [1,2], v2 = [2,5]$
  $v1 + v2 = [1,2] + [2,5] = [1+2, 2+5] = [3,7]$
- subtraction: Subtraction is just like addition but we compute the resultant vector of the addition of 1st vector with the negative of second vector:
  $\text{let } \ v1 = [1,2], v2 = [2,5]$
  $v1 - v2 = [1,2] - [2,5] = [1,2] + [-2,-5] = [1-2, 2-5] = [-1,-3]$
- Scalar multiplication: We use scalar multiplication to stretch a vector by the scalar quantity:
  $\text{let } \ v = [2,3]$
  $2v = 2[2,3] = [4,6]$

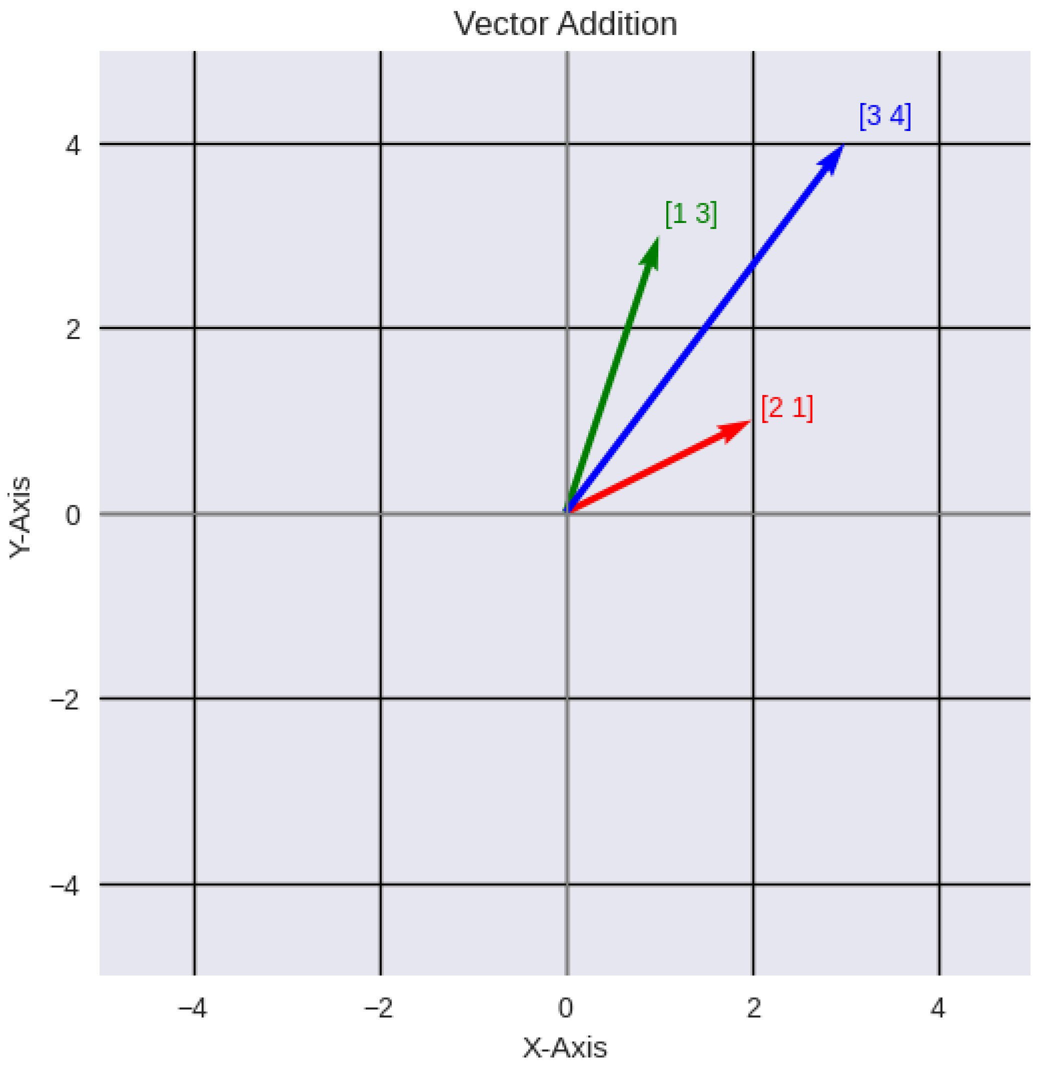
- Dot product: Dot product represents how much one vector points towards another vector.
  Positive dot product represents that a vector points towards another vector.
  Zero dot product represents that a vector points perpendicularly to another vector.
  Negative dot product represents that a vector points on the opposite direction to another vector.

- Line between vectors: It represents the angle between two vectors.

- Vector projection: Vector projection represents the image of one vector on another.

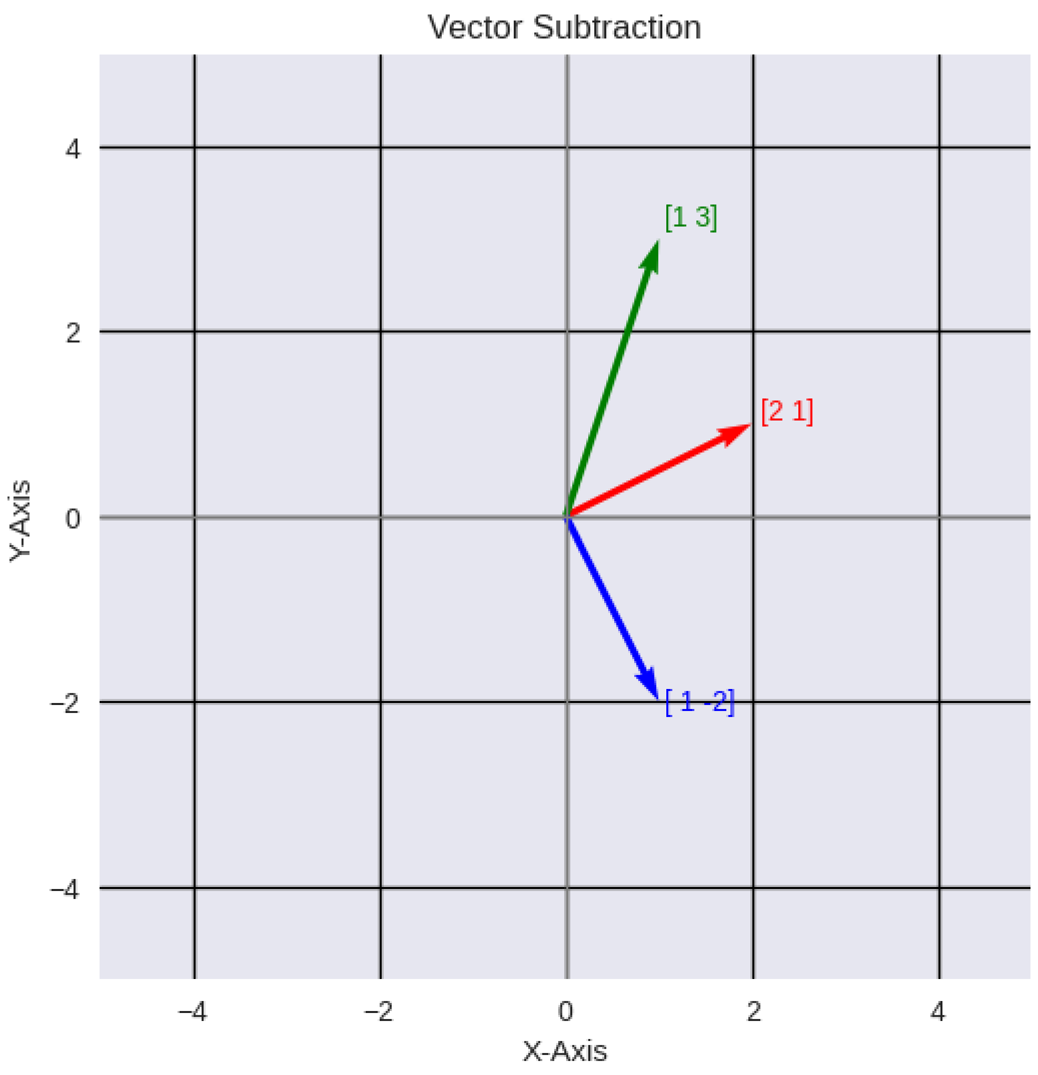
In [3]:
```
def vector_addition(a, b):
    print(f"a = {a}, b = {b}")
    print("Adding vectors:", a+b)
    plot_vectors([a, b, a+b], ['r', 'g', 'b'], "Vector Addition")

vector_addition(v1, v2)
```

```
a = [2 1], b = [1 3]
Adding vectors: [3 4]
```



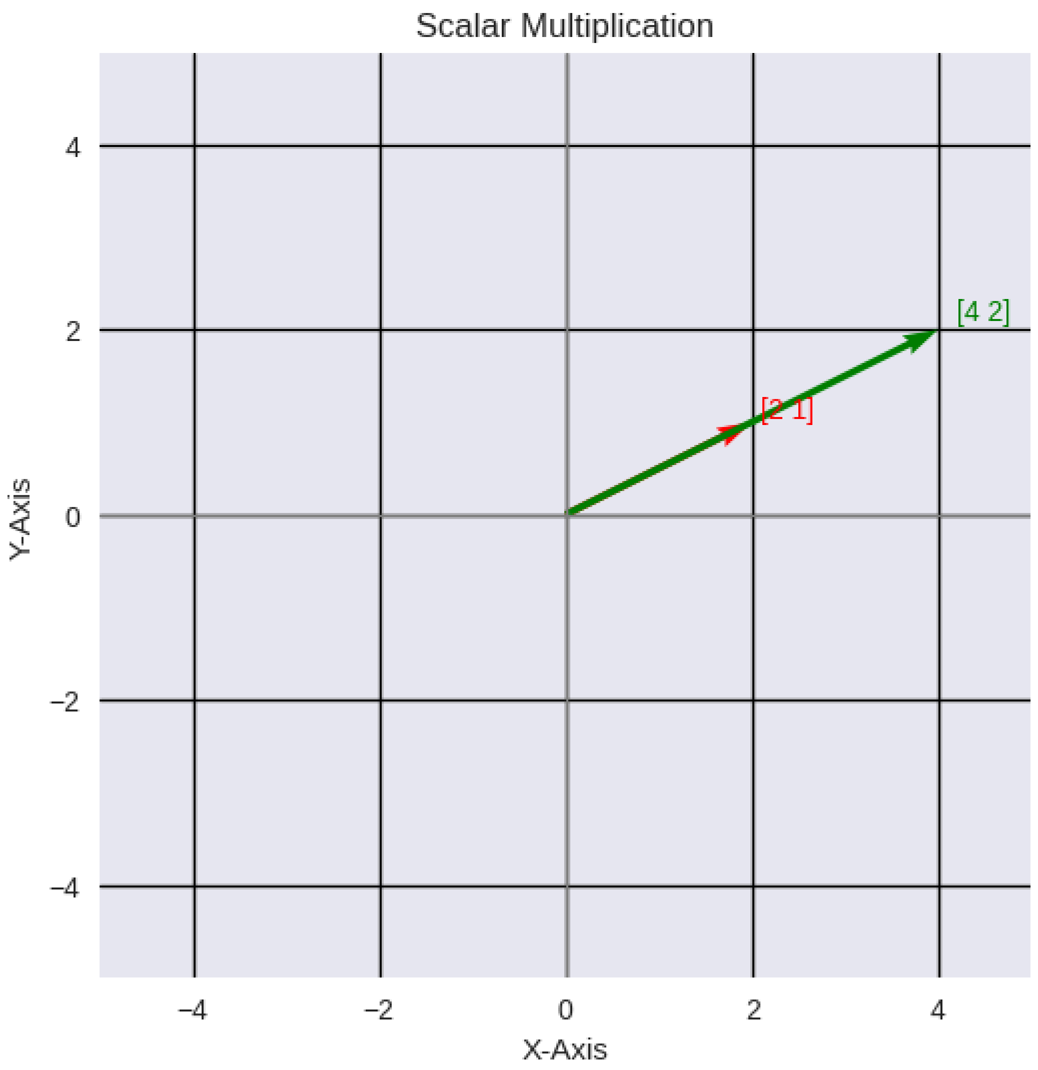
In [4]:
```
def vector_subtraction(a, b):
    print(f"Subtracting vectors:", a-b)
    plot_vectors([a, b, a-b], ['r','g','b'], "Vector Subtraction")

vector_subtraction(v1, v2)
```

```
Subtracting vectors: [ 1 -2]
```



In [5]:
```
def scalar_multiplication(a):
    print(f"Scalar multiplication:", 2 * a)
    plot_vectors([a, 2*a], ['r', 'g'], "Scalar Multiplication")

scalar_multiplication(v1)
```

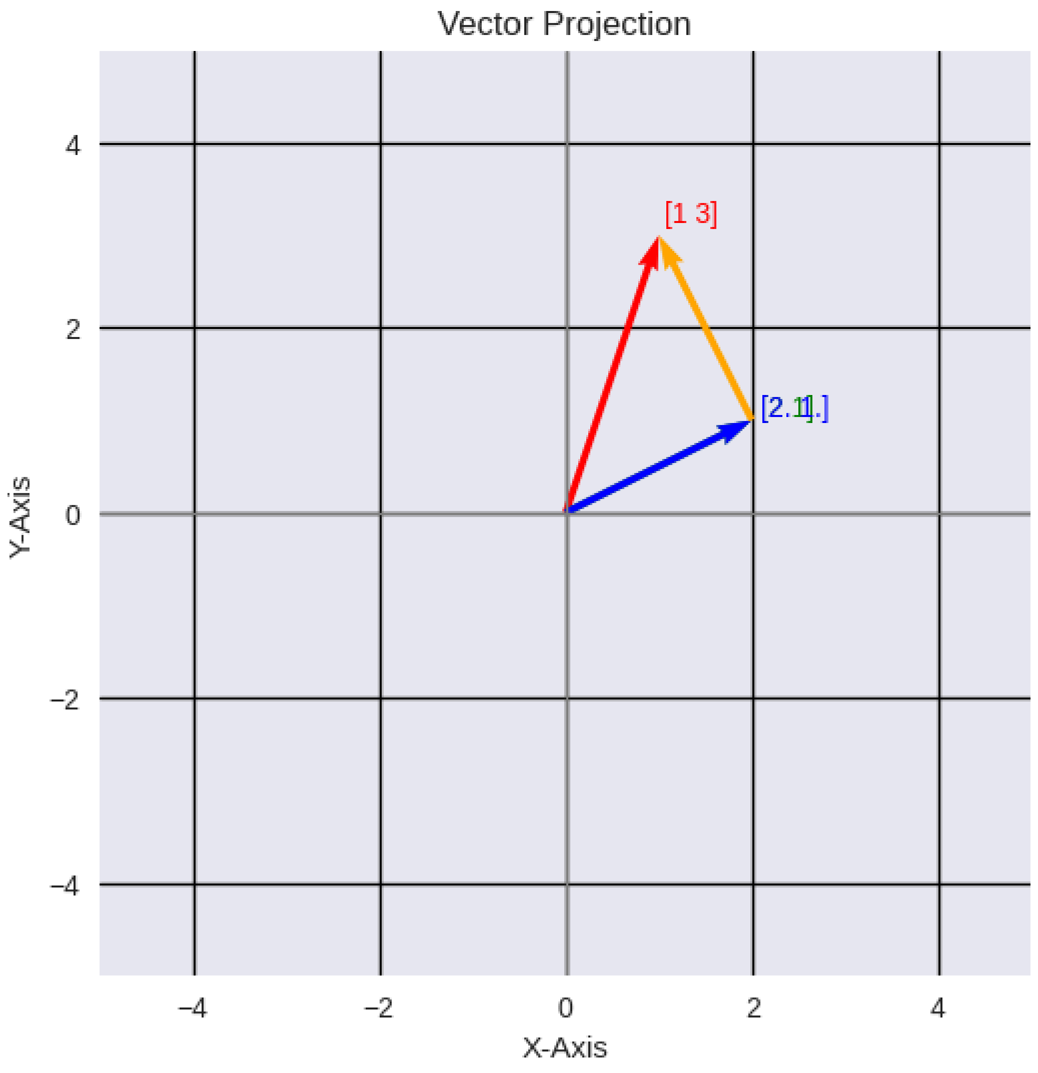
```
Scalar multiplication: [4 2]
```

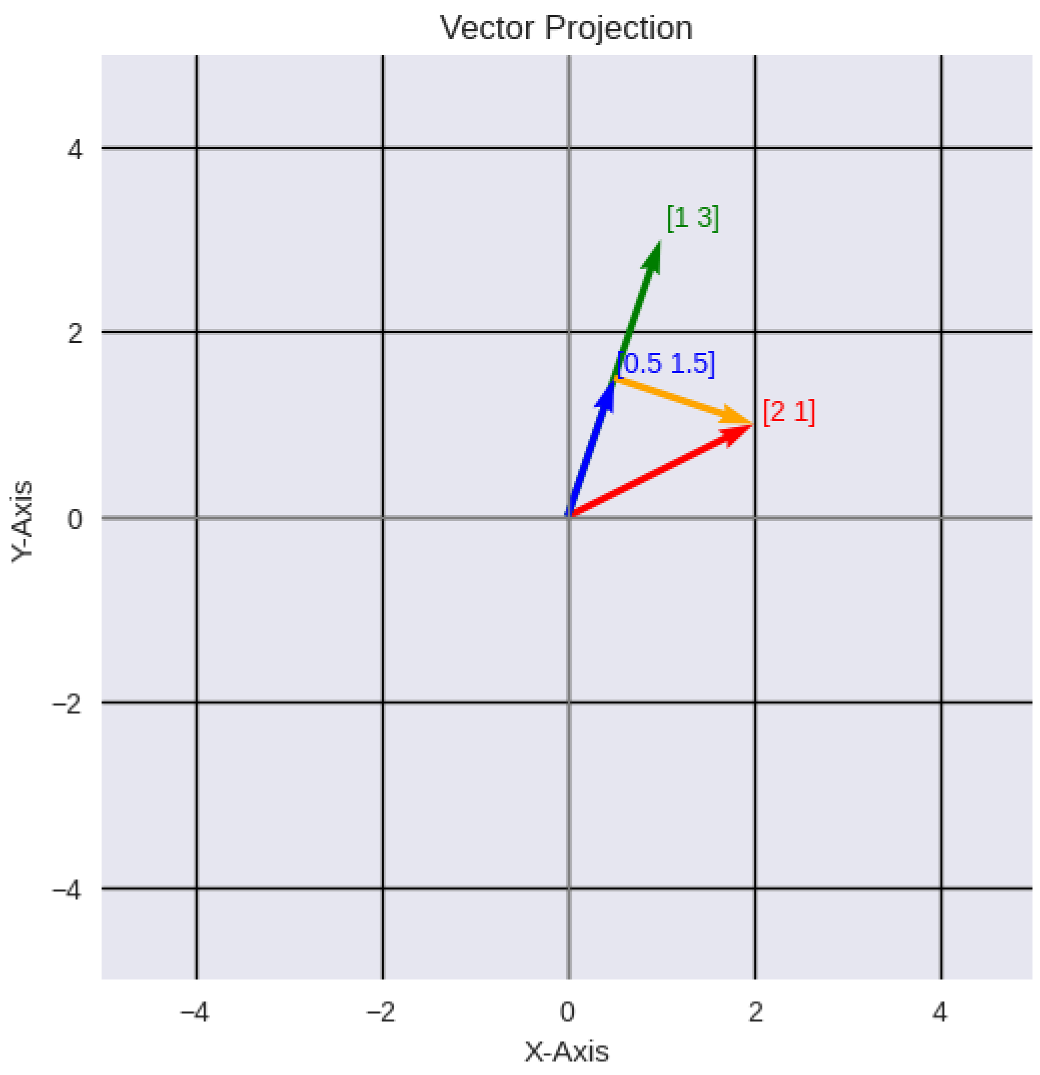


In [6]:
```python
def dot_product(a, b):
    print(f"a={a}, b={b}")
    mag_a, mag_b = np.linalg.norm(a).round(2), np.linalg.norm(b).round(2)
    print(f"magnitude of v1 = {mag_a}")
    print(f"magnitude of v2 = {mag_b}")
    dot = np.dot(a, b)
    print(f"Dot product = {dot}")
    angle = np.degrees(np.arccos(dot / (mag_a * mag_b))).round(2)
    print(f"Angle between two vectors: {angle} degrees")

dot_product(v1, v2)
```

```
a=[2 1], b=[1 3]
magnitude of v1 = 2.24
magnitude of v2 = 3.16
Dot product = 5
Angle between two vectors: 45.06 degrees
```

In [7]:
```python
def plot_projection(a, b):
    projection = (np.dot(a, b) / np.dot(b, b)).round(2) * b
    perpendicular = a - projection

    plt.figure(figsize=(6,6))
    plt.axhline(0, color='grey', lw=1)
    plt.axvline(0, color='grey', lw=1)
    plt.grid(True, linestyle='-', color='black')
    plt.quiver(0, 0, a[0], a[1], angles='xy', scale_units='xy', scale=1,
    plt.quiver(0, 0, b[0], b[1], angles='xy', scale_units='xy', scale=1,
    plt.quiver(0, 0, projection[0], projection[1], angles='xy', scale_uni
    plt.quiver(projection[0], projection[1], perpendicular[0], perpendicu

    plt.text(a[0]*1.05, a[1]*1.05, f"{a}", color='r', fontsize=10)
    plt.text(b[0]*1.05, b[1]*1.05, f"{b}", color='g', fontsize=10)
    plt.text(projection[0]*1.05, projection[1]*1.05, f"{projection}", col

    plt.xlim(-5,5)
    plt.ylim(-5,5)
    plt.xlabel("X-Axis")
    plt.ylabel("Y-Axis")
    plt.title("Vector Projection")
    plt.show()

plot_projection(v2, v1)
plot_projection(v1, v2)
```





Matrix Transformation

Here, we do Linear Transformations on matrices by multiplying them with others. Linear Transformation means, to keep the origin fixed and the grid lines should remain parallel and evenly spaced.

In the below example, the span of vectors between $(-2, 2)$ is taken in both axis.
Two "basis vectors" are defined which are $(1, 0)$ and $(0, 1)$.
After we apply Linear Transformation by multiplying it with a matrix by defining where the two "basis vectors" should land after transformation.
After Linear Transformation, we can know where any vector lands just by knowing where the two "basis vector" landed.

For eg:-
If the two basis vectors i and j are at $(1, 0)$ and $(0, 1)$ respectively, then:
And any vector v is at $2\hat{\imath} + 3\hat{\jmath}$, then

$$\begin{vmatrix} 1 & 2 \\ 2 & 1 \end{vmatrix}$$

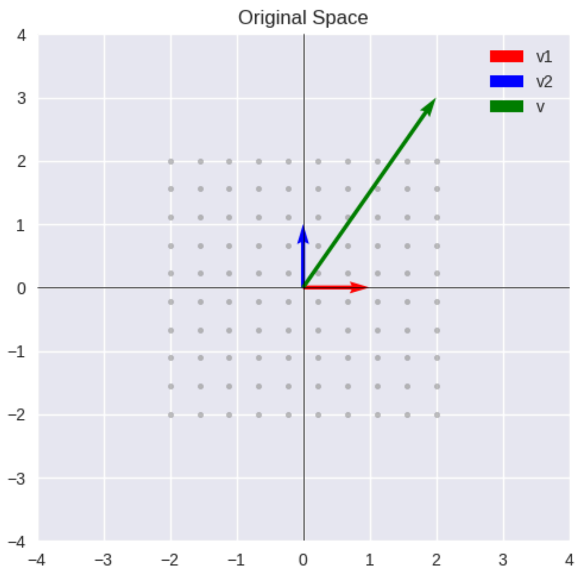
Now, we know that after transformation the basis vectors land at trans_i=(1,2) and trans_j=(2,1)

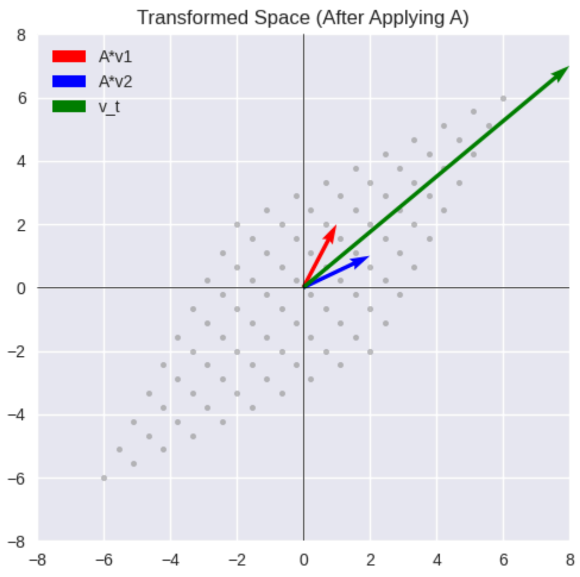
Now throught these transformed i and j vectors we can precisely know where the vector v landed

Thus, $v = 2.trans_{\hat{\imath}} + 3.trans_{\hat{\jmath}}$
$v = 2.(1,2) + 3.(2,1) = (2,4) + (6,3) = (8,7)$

thus we know, the transformed vector is at $(8,7)$.

In [9]:
```
def visualize_matrix(A):
    v1 = np.array([1, 0])
    v2 = np.array([0, 1])

    v = 2 * v1 + 3 * v2

    v1_t = A @ v1
    v2_t = A @ v2
    v_t = A @ v

    x = np.linspace(-2, 2, 10)
    y = np.linspace(-2, 2, 10)
    X, Y = np.meshgrid(x, y)
    grid_points = np.vstack([X.flatten(), Y.flatten()])
    transformed_grid = A @ grid_points

    fig, axes = plt.subplots(1, 2, figsize=(10, 5))

    axes[0].scatter(grid_points[0], grid_points[1], color='gray', s=10, a
    axes[0].quiver(0, 0, v1[0], v1[1], angles='xy', scale_units='xy', sca
    axes[0].quiver(0, 0, v2[0], v2[1], angles='xy', scale_units='xy', sca
    axes[0].quiver(0, 0, v[0], v[1], angles='xy', scale_units='xy', scale
    axes[0].set_title("Original Space")
    axes[0].set_xlim(-4, 4)
    axes[0].set_ylim(-4, 4)
    axes[0].grid(True)
    axes[0].axhline(0, color='black', linewidth=0.5)
    axes[0].axvline(0, color='black', linewidth=0.5)
    axes[0].legend()

    axes[1].scatter(transformed_grid[0], transformed_grid[1], color='gray
    axes[1].quiver(0, 0, v1_t[0], v1_t[1], angles='xy', scale_units='xy',
    axes[1].quiver(0, 0, v2_t[0], v2_t[1], angles='xy', scale_units='xy',
    axes[1].quiver(0, 0, v_t[0], v_t[1], angles='xy', scale_units='xy', s
    axes[1].set_title("Transformed Space (After Applying A)")
    axes[1].set_xlim(-8, 8)
    axes[1].set_ylim(-8, 8)
    axes[1].grid(True)
    axes[1].axhline(0, color='black', linewidth=0.5)
    axes[1].axvline(0, color='black', linewidth=0.5)
    axes[1].legend()

    plt.tight_layout()
    plt.show()

A = np.array([[1, 2],
              [2, 1]])

visualize_matrix(A)
```





Determinant

Determinant of a 2D matrix shows how much area of the unit square is spanned by the transformation.

In [13]:
```
A = np.array([[2, 3],
              [1, 4]])

det = np.linalg.det(A).round(2)
print(f"Determinant of A: {det}")
```

```
Determinant of A: 5.0
```