

# **Introduction to 2D Vector Rotation using MATLAB**

**Presented by: Ehsan Asali**

**02/13/2024**

# Introduction

- **Goal of the Lab**
  - To learn and apply matrix multiplication in rotating 2D vectors using MATLAB.
  - To visualize the rotation of vectors through scatter plots in MATLAB.



# Introduction

- **What You Will Achieve**
  - **Understanding Vector Rotation:** Grasp the concept of rotating vectors in a two-dimensional space around the origin and how this can be achieved using matrix operations.
  - **Practical MATLAB Application:** Apply matrix multiplication in a real-world scenario, enhancing your ability to use MATLAB for mathematical and engineering tasks.
  - **Enhanced MATLAB Skills:** Become familiar with essential MATLAB functions and constructs, including matrix operators, plotting functions, and graphical customization tools.

# Introduction

- **Importance of the Lab**
  - **Real-World Applications:** Vector rotation concepts are fundamental in various fields such as computer graphics, robotics, and physics simulations.
  - **MATLAB Proficiency:** MATLAB is a powerful tool for mathematical computations, data visualization, and algorithm development. Mastery of MATLAB functions and programming constructs is invaluable for engineers and scientists.



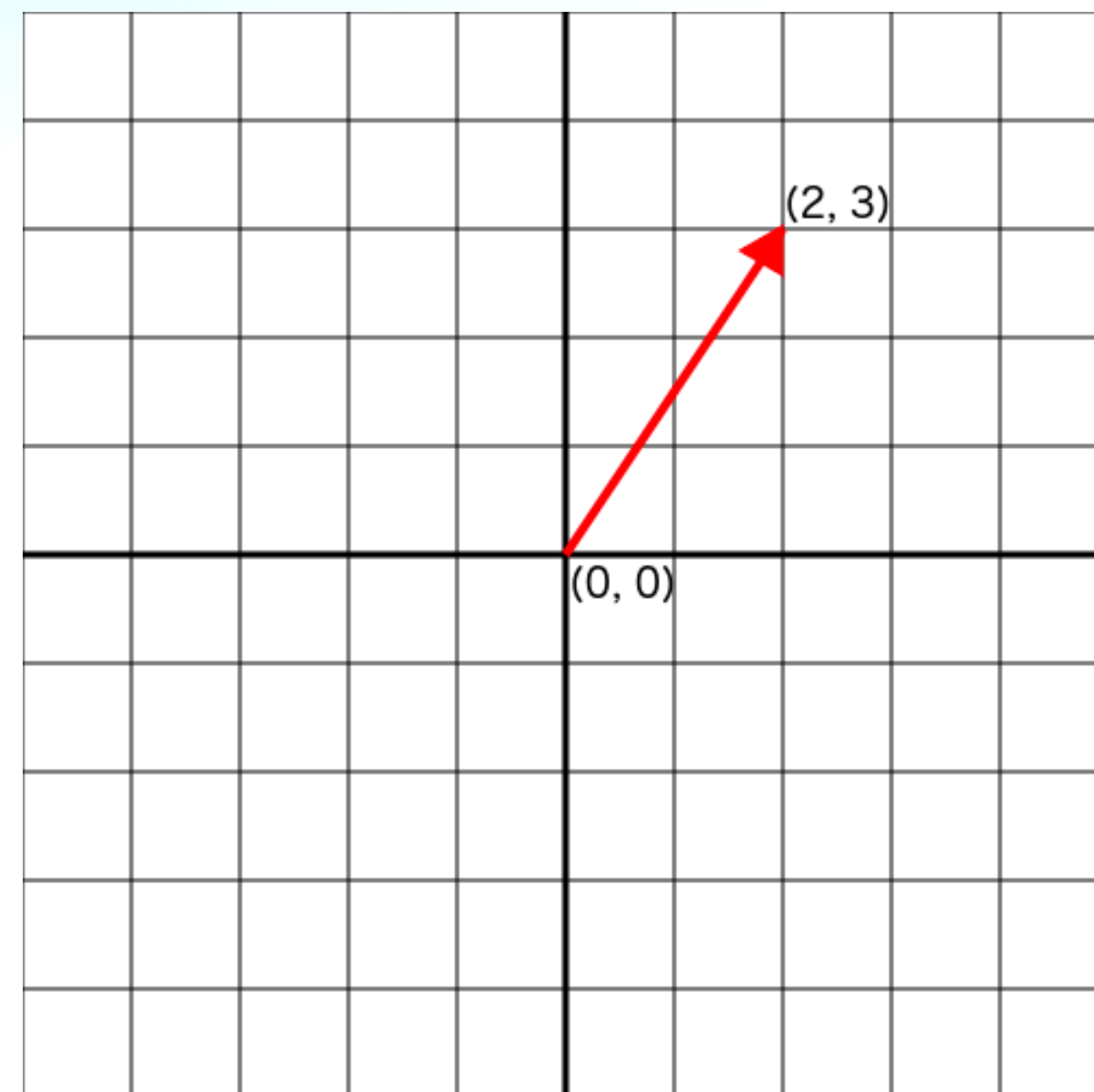
# Introduction

- **Lab Overview**
  - **Input:**
    - A matrix of 2D vectors  **$A$**
    - An angle  **$\theta$  (in degrees)**.
  - **Process:**
    - Rotate each vector in  $A$  by  $\theta$  degrees counterclockwise around the origin.
  - **Output:**
    - A new matrix of **rotated vectors  $B$**
    - **Scatter plots** of  $A$  and  $B$ .
  - **Expectations**
    - After completing this lab, you will be able to use MATLAB to manipulate matrices and plot data effectively. This skill set is crucial for both academic and professional success in STEM fields.

# Mathematical Background on Vectors and Matrices

- **Understanding Vectors**

- **Definition:** A vector in a two-dimensional space is represented as  $\mathbf{v} = (\mathbf{x}, \mathbf{y})$ , where  $x$  and  $y$  are the **horizontal** and **vertical** components of the vector, respectively.
- **Visualization:** Vectors can be visualized as arrows pointing from the origin  $(0,0)$  to the point  $(x,y)$  in a coordinate system.





# Mathematical Background on Vectors and Matrices

- **Matrices in Mathematics**

- **Definition:** A matrix is a rectangular array of numbers arranged in rows and columns. For our purposes, think of a matrix as a collection of 2D vectors.
- **Example:** A matrix **A** containing vectors **V1, V2, V3, ... Vn** can be represented as:

$$A = \begin{pmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_n & y_n \end{pmatrix}$$

# Mathematical Background on Vectors and Matrices

- **Rotation of 2D Vectors**
  - **Rotation Angle:** The angle  $\theta$  (in degrees or radians) by which we want to rotate the vector counterclockwise around the origin.
  - **Rotation Matrix:** A special matrix used to rotate vectors in the plane, defined for an angle  $\theta$  as:

$$R(\theta) = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}$$

- **Applying Rotation:** To rotate a vector  $v$ , we multiply it by the rotation matrix:

$$R(\theta): v' = R(\theta) \cdot v$$



# Mathematical Background on Vectors and Matrices

- **Why Matrix Multiplication?**
  - **Efficiency:** Allows us to apply the same rotation to a set of vectors (matrix) in a single operation.
  - **Uniformity:** Ensures that all vectors are rotated by the same angle, maintaining their relative positions.
- **Conversion Between Degrees and Radians**
  - Since MATLAB uses radians for trigonometric functions, it's crucial to convert angles from degrees to radians:

$$\text{radians} = \frac{\pi}{180} \times \text{degrees}$$

# Implementing Rotation in MATLAB

- The Rotation Matrix:

$$R(\theta) = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}$$

- Converting Degrees to Radians:

$$\text{radians} = \frac{\pi}{180} \times \text{degrees}$$

- Creating the Rotation Matrix in MATLAB
- Rotating a Vector
- Visualization



# MATLAB Functions for the Assignment

- **round:** Rounding off decimal values.
- **str2double:** Converting strings to double precision numbers.
- **scatter:** Creating scatter plots.
- **legend, xlabel, ylabel:** Adding legends and labels to plots.
- **axis:** Setting axis limits.
- **hold:** Holding the current plot to overlay additional plot elements.
- **xticks, yticks:** Customizing tick marks on axes.
- **exportgraphics:** Exporting plots as graphics files.

# MATLAB Functions for the Assignment

- **round**
  - **Purpose:** Rounds the elements of a matrix to the nearest integers.
  - **Usage:** Useful for cleaning up the output or for certain calculations where integer values are required.
  - **Syntax:** **B = round(A);**
    - **A** is the input matrix or vector
    - **B** is the rounded result.


```
% Define the original matrix with floating-point numbers
A = [1.2, 3.5; 4.8, 6.1];

% Round all elements to the nearest integer
B = round(A);

% Display the rounded matrix
disp('Rounded to nearest integer:');
disp(B);

% Round all elements to one decimal place
C = round(A, 1);

% Display the matrix rounded to one decimal place
disp('Rounded to one decimal place:');
disp(C);
```





# MATLAB Functions for the Assignment

- **str2double**
  - **Purpose:** Converts strings to double precision floating point numbers.
  - **Usage:** Essential when reading numerical values from text inputs or files that are initially recognized as strings.
  - **Syntax:** `nums = str2double(strs);`
    - **strs** can be a string array, character vector, or cell array of character vectors.
    - **nums** is the numeric representation.

# MATLAB Functions for the Assignment

- **scatter**
  - **Purpose:** Creates a scatter plot of y vs. x with varied marker sizes and colors.
  - **Usage:** Ideal for visualizing the original and rotated vectors on a 2D plane.
  - **Syntax:** `scatter(x,y);`
    - **x** and **y** are vectors containing the x-coordinates and y-coordinates of the points to be plotted.



# MATLAB Functions for the Assignment

- **legend**
  - **Purpose:** Adds a legend to the axes or chart.
  - **Usage:** Useful for distinguishing between different data sets plotted on the same graph, such as original vs. rotated vectors.
  - **Syntax:** `legend('label1','label2',...);`
    - Labels correspond to the plotted data series.

# MATLAB Functions for the Assignment

- **xlabel, ylabel**
  - **Purpose:** Sets the x-axis label and y-axis label of the current axes.
  - **Usage:** Essential for labeling axes to indicate what data is being plotted.
  - **Syntax:**
    - **xlabel**('X-axis label');
    - **ylabel**('Y-axis label');



# MATLAB Functions for the Assignment

- **axis**
  - **Purpose:** Sets the limits for the axes of the current plot.
  - **Usage:** Useful for defining a specific viewing window for your data, especially when keeping the -5 to 5 range for both axes.
  - **Syntax:** `axis([xmin xmax ymin ymax]);`

# MATLAB Functions for the Assignment

- **hold**
  - **Purpose:** Holds the current plot and all axis properties so that subsequent graphing commands add to the existing graph.
  - **Usage:** Necessary for plotting multiple datasets on the same figure without clearing the previous data.
  - **Syntax:**
    - **hold on;** to add to the existing plot.
    - **hold off;** to release the hold.



# MATLAB Functions for the Assignment

- **xticks, yticks**
  - **Purpose:** Sets the locations of the x-axis and y-axis tick marks.
  - **Usage:** Customizes the appearance of the axes, allowing for specific tick mark placement.
  - **Syntax:**
    - **xticks**([x1 x2 x3 ...]);
    - **yticks**([y1 y2 y3 ...]);

# MATLAB Functions for the Assignment

- **exportgraphics**
  - **Purpose:** Exports graphics to a file.
  - **Usage:** Ideal for saving your plots as high-quality images for reports or presentations.
  - **Syntax:** `exportgraphics(gca, 'filename.png');`
    - `gca` gets the current axes, and `'filename.png'` specifies the output file name and format.



# Plotting in MATLAB

- **Basic Plotting with *plot* function**
  - **Purpose:** Plot  $y$  versus  $x$  as lines and/or markers.
  - **Syntax:** `plot(x, y)`
  - **Example:**

```
x = linspace(0, 2*pi, 100);  
y = sin(x);  
plot(x, y);  
title('Sine Wave');  
xlabel('x');  
ylabel('sin(x)');
```

# Plotting in MATLAB

- **Scatter Plots with *scatter***
  - **Purpose:** Create a scatter plot with circular markers.
  - **Syntax:** `scatter(x, y)`
  - **Example:**

```
x = randn(1, 100);  
y = randn(1, 100);  
scatter(x, y);  
title('Random Scatter Plot');  
xlabel('x');  
ylabel('y');
```



# Plotting in MATLAB

- **Customizing Plots**

- **Axis Limits:** Control the range of the axes using `xlim` and `ylim`.

```
xlim([0, 2*pi]);  
ylim([-1, 1]);
```

- **Markers and Line Styles:** Customize the appearance of plots.

```
plot(x, y, 'r--', 'LineWidth', 2);
```

- **Adding Legends and Grids:** Enhance readability.

```
legend('Sine Function');  
grid on;
```

# Plotting in MATLAB

- **Advanced Plotting**
  - **Multiple Plots on the Same Axes:** Use hold on and hold off.
- **Subplots:** Display multiple plots in a single figure.
- **Saving Plots**
  - Using **saveas** and **exportgraphics**

```
plot(x, sin(x));  
hold on;  
plot(x, cos(x), 'r');  
legend('sin(x)', 'cos(x)');  
hold off;
```

```
subplot(2,1,1); % 2 rows, 1 column, 1st subplot  
plot(x, sin(x));  
title('Sine Wave');  
  
subplot(2,1,2); % 2nd subplot  
plot(x, cos(x), 'r');  
title('Cosine Wave');
```

```
saveas(gcf, 'sine_wave.png');  
exportgraphics(gca, 'cosine_wave.pdf', 'ContentType', 'vector');
```