Foundations of Linear Algebra for Robotics

Prof. Gerardo Flores

January 2025

Robotics and Automation (CSCE 3345 - SENG 3340)

TAMIU

Mathematical Foundations

1. **Set of Real Numbers**:
   - $\mathbb{R}$: Denotes the set of real numbers.
   - $\mathbb{R}^n$: Represents the vector space of n-tuples over $\mathbb{R}$.
2. **Notations**:
   - **Scalars**: Represented by lowercase letters $a, b, c, x, y, \ldots$ (e.g., $a \in \mathbb{R}$).
   - **Vectors**: Represented as column vectors. For $x \in \mathbb{R}^n$:

     $$x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \quad x_i \in \mathbb{R}.$$

     Alternatively:

     $$x = [x_1, x_2, \ldots, x_n]^T,$$

     where T denotes the transpose.
   - **Matrices**: Represented by uppercase letters $A, B, C, R, \ldots$.

Mathematical Foundations

3. **Vector Norm**:

   - The length or norm of a vector $x \in \mathbb{R}^n$ is given by:

$$\|x\| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}.$$

Python Example

```
import numpy as np

# Define the vector
X = np.array([-2, 3, -1])

# Compute the norm of the vector
norm_X = np.linalg.norm(X)

# Output the result
print(f"The norm of X is: {norm_X:.4f}")
```

MATRICES (Notation)

1. **Definition**:

    - A real $m \times n$ matrix is represented as:

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix}, \quad A \in \mathbb{R}^{m \times n}.$$

    - Where m is the number of rows and n is the number of columns.
    - If $m = n$, the matrix is square.

Matrix Multiplication

- Two matrices A and B can be multiplied if the number of columns in A equals the number of rows in B.
- If A is $m \times n$ and B is $n \times p$, the resulting matrix C will be $m \times p$.
- The entry in row i and column j of C is computed as:

$$C[i,j] = \sum_{k=1}^{n} A[i,k] \cdot B[k,j]$$

Example:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

$$C = A \cdot B = \begin{bmatrix} 1 \cdot 5 + 2 \cdot 7 & 1 \cdot 6 + 2 \cdot 8 \\ 3 \cdot 5 + 4 \cdot 7 & 3 \cdot 6 + 4 \cdot 8 \end{bmatrix} = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

```python
import numpy as np

# Define matrices A and B
A = np.array([[1, 2],
              [3, 4]])
B = np.array([[5, 6],
              [7, 8]])

# Perform matrix multiplication
C = np.dot(A, B)

# Print the result
print("Matrix A:")
print(A)
print("\nMatrix B:")
print(B)
print("\nMatrix Multiplication Result (C = A * B):")
print(C)
```

Matrix-Vector Multiplication

- A matrix A can be multiplied by a vector $\mathbf{v}$ if the number of columns in A equals the number of entries in $\mathbf{v}$.
- If A is $m \times n$ and $\mathbf{v}$ is $n \times 1$, the resulting vector $\mathbf{u}$ will be $m \times 1$.
- The i-th entry of $\mathbf{u}$ is computed as:

$$u[i] = \sum_{k=1}^{n} A[i,k] \cdot v[k]$$

Example:

$$A = \begin{bmatrix} 2 & 1 \\ 0 & 3 \end{bmatrix}, \quad \mathbf{v} = \begin{bmatrix} 4 \\ 5 \end{bmatrix}$$

$$\mathbf{u} = A \cdot \mathbf{v} = \begin{bmatrix} 2 \cdot 4 + 1 \cdot 5 \\ 0 \cdot 4 + 3 \cdot 5 \end{bmatrix} = \begin{bmatrix} 13 \\ 15 \end{bmatrix}$$

```python
import numpy as np

# Define matrix A and vector v
A = np.array([[2, 1],
              [0, 3]])
v = np.array([4, 5])

# Perform matrix-vector multiplication
u = np.dot(A, v)

# Print the result
print("Matrix A:")
print(A)
print("\nVector v:")
print(v)
print("\nMatrix-Vector Multiplication Result (u = A * v):")
print(u)
```

Scalar Product

1. **Definition**:
   - The scalar product (dot product) of two vectors $x, y \in \mathbb{R}^n$ is defined as:

$$\langle x, y \rangle = x^T y = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n.$$

2. **Properties**:
   - **Norm Relation**: The norm of a vector can be derived from the scalar product:

$$\|x\| = \sqrt{\langle x, x \rangle}.$$

   - **Commutativity**:

$$\langle x, y \rangle = \langle y, x \rangle.$$

3. **Useful Inequalities**:
   - **Cauchy-Schwarz**:

$$|\langle x, y \rangle| \leq \|x\| \|y\|.$$

   - **Triangle Inequality**:

$$\|x + y\| \leq \|x\| + \|y\|.$$

Scalar Product

vector_analysis.py

4. **Geometric Interpretation** (for vectors in $\mathbb{R}^3$):

- The scalar product can also be expressed as:

$$\langle x, y \rangle = \|x\| \|y\| \cos(\theta),$$

where θ is the angle between x and y.

```python
import numpy as np

# Define two vectors
x = np.array([1, 2, 3])
y = np.array([4, 5, 6])

# Compute the dot (scalar) product
dot_product = np.dot(x, y)
print(f"Dot product (x, y): {dot_product}")

# Compute the norms of x and y
norm_x = np.linalg.norm(x)
norm_y = np.linalg.norm(y)
print(f"||x||: {norm_x:.4f}")
print(f"||y||: {norm_y:.4f}")

# Verify Cauchy-Schwarz inequality
cauchy_schwarz = abs(dot_product) <= norm_x * norm_y
print(f"Cauchy-Schwarz inequality holds: {cauchy_schwarz}")

# Verify Triangle inequality
triangle_inequality = np.linalg.norm(x + y) <= norm_x + norm_y
print(f"Triangle inequality holds: {triangle_inequality}")

# Geometric interpretation (cosine of the angle)
cos_theta = dot_product / (norm_x * norm_y)
theta = np.arccos(cos_theta)  # Angle in radians
print(f"cos(θ): {cos_theta:.4f}")
print(f"Angle θ (in degrees): {np.degrees(theta):.2f}")
```

Trace of a Matrix

The **trace** of a matrix A is defined as:

$$\operatorname{tr}(A) = \sum_{i=1}^{n} A_{ii}$$

where A_{ii} represents the diagonal elements of the matrix.

Additionally, the trace can also be expressed as the sum of the eigenvalues λ_i of the matrix:

$$\operatorname{Tr}(A) = \sum_{i=1}^{n} \lambda_i$$

Exercise

Find the trace of the matrix

Given the matrix A:

$$A = \begin{bmatrix} 5 & 3 & 5 \\ 4 & -1 & 2 \\ -3 & 8 & 7 \end{bmatrix}$$

The trace is defined as the sum of its diagonal elements:

$$\operatorname{tr}(A) = 5 + (-1) + 7 = 11$$

```python
import numpy as np

# Define the matrix
A = np.array([[5, 3, 5],
              [4, -1, 2],
              [-3, 8, 7]])

# Calculate the trace
trace_A = np.trace(A)
print("Trace of matrix A:", trace_A)
```

Answer:

$$\text{Trace}(A) = 11$$

Exercises

1. **Find the dot product**

   Compute the dot product of two vectors:

$$a = \begin{bmatrix} -12 \\ 2 \\ -2 \end{bmatrix}, \quad b = \begin{bmatrix} -14 \\ 10 \\ -4 \end{bmatrix}$$

2. **Verify the norm relation**

   Verify that the norm of a vector can be derived from the dot product:

$$||a|| = \sqrt{\langle a, a \rangle}$$

   For $a = \begin{bmatrix} -12 \\ 2 \\ -2 \end{bmatrix}$, calculate $||a||$.

```python
import numpy as np

# Define vectors
a = np.array([-12, 2, -2])
b = np.array([-14, 10, -4])

# Calculate the dot product
dot_product = np.dot(a, b)
print("Dot Product:", dot_product)
```

Answer:

$$\text{Dot Product} = 160$$

```python
# Norm of vector a
norm_a = np.sqrt(np.dot(a, a))
print("Norm of vector a:", norm_a)
```

Answer:

$$||a|| = 12.3288$$

Exercises

Check the Cauchy-Schwarz inequality

Confirm that:

$$|\langle a, b\rangle| \leq ||a|| \cdot ||b||$$

```
# Norm of vector b
norm_b = np.sqrt(np.dot(b, b))

# Cauchy-Schwarz inequality
left_side = abs(np.dot(a, b))
right_side = norm_a * norm_b
print("Cauchy-Schwarz inequality holds:", left_side <= right_side)
```

Answer:

The inequality holds because:

$$|\langle a, b\rangle| = 160, \quad ||a|| \cdot ||b|| = 169.651$$

Expressing Vectors as Linear Combinations of Unit Vectors

We define the standard unit vectors in $\mathbb{R}^3$ as follows:

$$\mathbf{i} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{j} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad \mathbf{k} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

Using this notation, any vector $\mathbf{x}$ in $\mathbb{R}^3$, represented as:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix},$$

can also be written as:

$$\mathbf{x} = x_1\mathbf{i} + x_2\mathbf{j} + x_3\mathbf{k}.$$

```python
import numpy as np

# Define the unit vectors
i = np.array([1, 0, 0])
j = np.array([0, 1, 0])
k = np.array([0, 0, 1])

# Define the vector x
x = np.array([2, -3, 4])

# Express x as a combination of i, j, k
x1, x2, x3 = x
linear_combination = x1 * i + x2 * j + x3 * k

# Print the result
print("Vector x:", x)
print("Linear combination: {}i + {}j + {}k".format(x1, x2, x3))
```

```
Vector x: [ 2 -3  4]
Linear combination: 2i + -3j + 4k
```

Vector Product or Cross Product

1. **Definition:** The vector product (cross product) of two vectors x and y in $\mathbb{R}^3$ is a vector c, defined as:

$$c = x \times y = \det \begin{bmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{bmatrix}$$

   Expanding this determinant, the result is:

$$c = (x_2y_3 - x_3y_2)\mathbf{i} + (x_3y_1 - x_1y_3)\mathbf{j} + (x_1y_2 - x_2y_1)\mathbf{k}.$$

2. **Magnitude of the Cross Product:** The magnitude of the cross product $\|c\|$ is given by:

$$\|c\| = \|x\|\|y\|\sin(\theta),$$

   where θ is the angle between the vectors x and y.

3. **Direction:** The direction of the vector c is determined by the **right-hand rule**.
   - Curl the fingers of your right hand from vector x towards vector y, and your thumb will point in the direction of c.

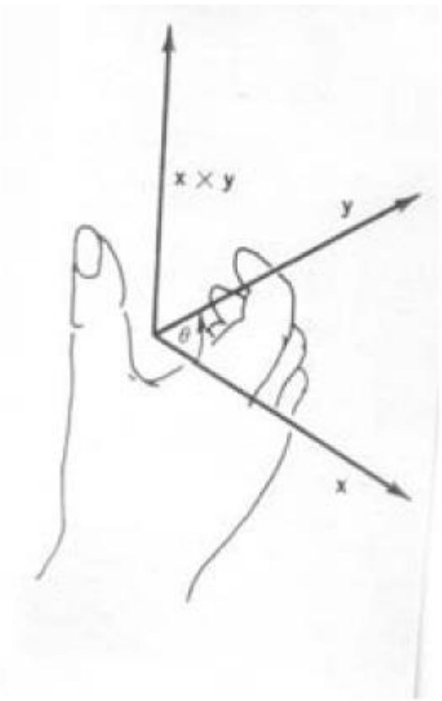
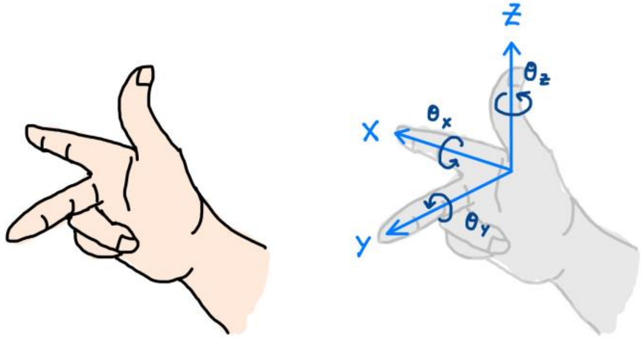


Fig. B.1 The right hand rule.



4. Properties of the Cross Product:

1. **Anticommutativity**:

$$x \times y = -y \times x$$

2. **Distributivity**:

$$x \times (y + z) = x \times y + x \times z$$

3. **Scalar Multiplication**:

$$\alpha(x \times y) = (\alpha x) \times y = x \times (\alpha y)$$

These properties highlight key algebraic behaviors of the cross product.

Exercises

Given:

Two vectors $\mathbf{u} = 3\mathbf{i} - \mathbf{j} + \mathbf{k}$ and $\mathbf{v} = 2\mathbf{i} - 3\mathbf{j} + \mathbf{k}$.

Tasks:

1. Find the cross product $\mathbf{u} \times \mathbf{v}$ and verify that this vector is orthogonal to both $\mathbf{u}$ and $\mathbf{v}$.
2. Compute $\mathbf{v} \times \mathbf{u}$ and compare it with $\mathbf{u} \times \mathbf{v}$.

---

1. **Computing the cross product $\mathbf{u} \times \mathbf{v}$:**

$$\mathbf{u} \times \mathbf{v} = (2, -1, -7)$$

2. **Verifying orthogonality:**
   - $\mathbf{u} \cdot (\mathbf{u} \times \mathbf{v}) = 0$
   - $\mathbf{v} \cdot (\mathbf{u} \times \mathbf{v}) = 0$

   This confirms that the resulting vector is orthogonal to both $\mathbf{u}$ and $\mathbf{v}$.

3. **Computing the cross product $\mathbf{v} \times \mathbf{u}$:**

$$\mathbf{v} \times \mathbf{u} = (-2, 1, 7)$$

We observe that $\mathbf{v} \times \mathbf{u} = -(\mathbf{u} \times \mathbf{v})$, which is consistent with the antisymmetric property of the cross product.

cross_product_verification.py

```python
import numpy as np

# Define vectors u and v
u = np.array([3, -1, 1])
v = np.array([2, -3, 1])

# Compute the cross product u × v
cross_u_v = np.cross(u, v)

# Compute the cross product v × u
cross_v_u = np.cross(v, u)

# Verify orthogonality: dot product should be 0
# If u × v is orthogonal to both u and v, then their dot product should be zero
dot_u_cross = np.dot(u, cross_u_v)
dot_v_cross = np.dot(v, cross_u_v)

# Print results
print("Cross product u × v:", cross_u_v)
print("Cross product v × u:", cross_v_u)
print("Dot product u · (u × v):", dot_u_cross)
print("Dot product v · (u × v):", dot_v_cross)
```

Visual Studio Code installation

Visual Studio Code

https://code.visualstudio.com/docs/setup/mac [choose your operating system]

https://code.visualstudio.com/docs/python/python-tutorial Python tutorial for visual studio code including virtual environment setup plot_3d_vectors.py

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

# Define the vectors a, b, and their sum c
a = np.array([4, -2, 1])  # Vector a
b = np.array([1, -1, 3])  # Vector b
c = a + b                 # Vector c as the sum of a and b

# Define the starting points of the vectors (all starting at the origin)
starts = np.zeros((3, 3))  # A 3x3 array of zeros representing the origin [0, 0, 0] for all vectors

# Define the endpoints of the vectors
ends = np.array([a, b, c])  # Each row corresponds to the endpoint of a, b, and c

# Create a 3D plot
fig = plt.figure()  # Initialize a new figure for the plot
ax = fig.add_subplot(111, projection='3d')  # Add a 3D subplot to the figure

# Plot the vectors using quiver
ax.quiver(
    starts[:, 0], starts[:, 1], starts[:, 2],  # Starting points of the vectors (all at the origin)
    ends[:, 0], ends[:, 1], ends[:, 2],        # Endpoints of the vectors (defined by vectors a, b, and c)
    color=['r', 'g', 'b'],                     # Colors for each vector (red, green, blue)
    arrow_length_ratio=0.1                     # Adjust the arrow size relative to the vector length
)

# Set axis labels for better understanding of the plot
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_zlabel('Z-axis')

# Set the aspect ratio to be equal for better visualization
ax.set_box_aspect([1, 1, 1])  # Make the axes scale equally

# Show the plot
plt.show()
```

Orthogonal and Orthonormal Vectors

Orthogonal Vectors

- Two vectors $\mathbf{u}$ and $\mathbf{v}$ are orthogonal if their dot product is zero:

$$\mathbf{u} \cdot \mathbf{v} = 0$$

- In three-space, multiple vectors can be mutually perpendicular.

Orthonormal Vectors

- Vectors that are both **orthogonal** and **unit vectors** are called orthonormal.
- Example: If $||\mathbf{u}|| = 1$, $||\mathbf{v}|| = 1$, and $\mathbf{u} \cdot \mathbf{v} = 0$, then $\mathbf{u}$ and $\mathbf{v}$ are orthonormal.

Orthogonal Matrix

- A square matrix Q is orthogonal if:

$$Q^\top Q = QQ^\top = I$$

  where I is the identity matrix.

- **Properties**:
  - The transpose of an orthogonal matrix equals its inverse:

$$Q^\top = Q^{-1}$$

Examples of Orthogonal Matrices

1. **Identity Transformation**:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

2. **Rotation Matrix** (e.g., rotation by 16.26°):

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

   For $\theta = 16.26^\circ$:

$$R = \begin{bmatrix} 0.96 & -0.28 \\ 0.28 & 0.96 \end{bmatrix}$$

3. **Reflection Across the x-axis**:

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

4. **Permutation of Coordinate Axes**:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Differentiation of Vectors

1. **Definition**

   Suppose the vector $x(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_n(t) \end{bmatrix}$ is a function of time. Then, the time derivative $\dot{x}$ of x is defined as:

$$\dot{x} = \begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \\ \vdots \\ \dot{x}_n(t) \end{bmatrix}.$$

2. **Coordinate-Wise Differentiation**
   - The derivative $\frac{dA}{dt}$ of a matrix $A = (a_{ij})$ is just the matrix $\dot{A} = (\dot{a}_{ij})$.
   - Similar statements hold for integration of vectors and matrices.

3. **Product Rules**

   Scalar and vector products satisfy product rules similar to differentiation of ordinary functions:

   - **Dot Product Rule:**

$$\frac{d}{dt}\langle x, y\rangle = \langle \frac{dx}{dt}, y\rangle + \langle x, \frac{dy}{dt}\rangle.$$

   - **Cross Product Rule:**

$$\frac{d}{dt}(x \times y) = \frac{dx}{dt} \times y + x \times \frac{dy}{dt}.$$

This provides the rules for differentiating scalar and vector products, ensuring consistency in their application.

```python
import sympy as sp  # Import SymPy for symbolic computation

# Define the symbolic variable
x = sp.Symbol('x')

# Define the vector as a column matrix
X = sp.Matrix([
    sp.sin(x),        # First component: sin(x)
    x**2,             # Second component: x^2
    sp.cos(x) - 3*x   # Third component: cos(x) - 3x
])

# Compute the derivative of the vector with respect to x
dX_dx = X.diff(x)

# Display the result in a readable format
sp.pprint(dX_dx)
```

symbol_derivative.py

Linear Dependence and Independence

1. **Linear Dependence**

   A set of vectors $\{\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_k\}$ in a vector space V is linearly dependent if there exist scalars $a_1, a_2, \ldots, a_k$, not all zero, such that:

   $$a_1\vec{v}_1 + a_2\vec{v}_2 + \cdots + a_k\vec{v}_k = \vec{0}.$$

   This means at least one vector in the set can be written as a combination of the others.

2. **Linear Independence**

   A set of vectors $\{\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_n\}$ is linearly independent if the equation:

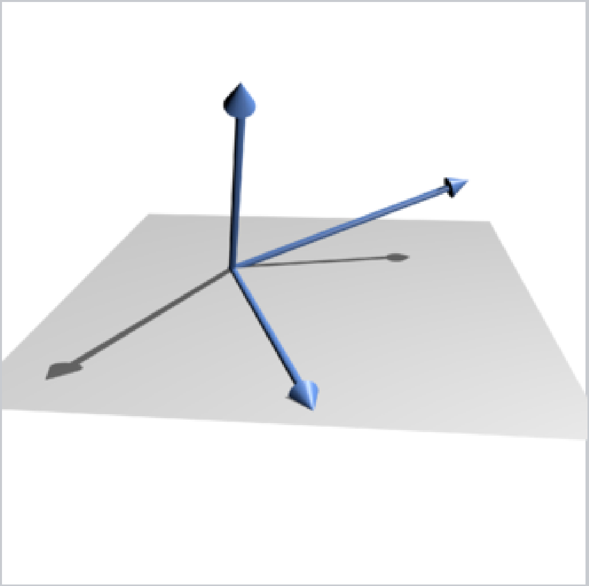
   $$a_1\vec{v}_1 + a_2\vec{v}_2 + \cdots + a_n\vec{v}_n = \vec{0}$$

   holds **only** when all the scalars $a_1, a_2, \ldots, a_n$ are zero.

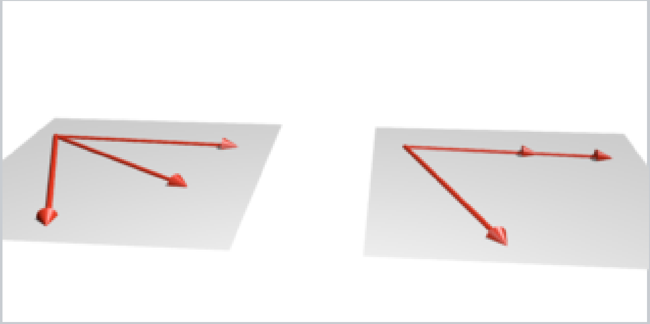
   This means none of the vectors can be expressed as a combination of the others.

Simplified Explanation:

- If you can express one vector in the set as a mix of the others, the vectors are **dependent**.
- If no vector can be made using the others, they are **independent**.



Linearly independent vectors in $\mathbb{R}^3$



Linearly dependent vectors in a plane in $\mathbb{R}^3$.

Exercise

Problem: Matrix Reduction Using Gauss-Jordan Elimination

Given the following **augmented matrix**:

$$\begin{bmatrix} 1 & 3 & 1 & 9 \\ 1 & 1 & -1 & 1 \\ 3 & 11 & 5 & 35 \end{bmatrix}$$

The goal is to transform it into its **reduced row echelon form (RREF)** using the **Gauss-Jordan elimination method**.

Step 1: Make the pivot in the first column a `1`.

The pivot element in the first column is already `1` (in row 1). No operation is needed here.

Step 2: Eliminate the other elements in the first column (make them `0`).

- Subtract **row 1** from **row 2** ($R_2 \to R_2 - R_1$):

$$R_2 = [1, 1, -1, 1] - [1, 3, 1, 9] = [0, -2, -2, -8]$$

- Subtract **3 × row 1** from **row 3** ($R_3 \to R_3 - 3R_1$):

$$R_3 = [3, 11, 5, 35] - 3 \cdot [1, 3, 1, 9] = [0, 2, 2, 8]$$

Updated matrix:

$$\begin{bmatrix} 1 & 3 & 1 & 9 \\ 0 & -2 & -2 & -8 \\ 0 & 2 & 2 & 8 \end{bmatrix}$$

Step 3: Make the pivot in the second column a `1`.

- Divide **row 2** by -2 ($R_2 \to R_2 / -2$):

$$R_2 = [0, -2, -2, -8] / -2 = [0, 1, 1, 4]$$

Updated matrix:

$$\begin{bmatrix} 1 & 3 & 1 & 9 \\ 0 & 1 & 1 & 4 \\ 0 & 2 & 2 & 8 \end{bmatrix}$$

Step 4: Eliminate the other elements in the second column (make them `0`).

- Subtract **3 × row 2** from **row 1** ($R_1 \to R_1 - 3R_2$):

$$R_1 = [1, 3, 1, 9] - 3 \cdot [0, 1, 1, 4] = [1, 0, -2, -3]$$

- Subtract **2 × row 2** from **row 3** ($R_3 \to R_3 - 2R_2$):

$$R_3 = [0, 2, 2, 8] - 2 \cdot [0, 1, 1, 4] = [0, 0, 0, 0]$$

Updated matrix:

$$\begin{bmatrix} 1 & 0 & -2 & -3 \\ 0 & 1 & 1 & 4 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Step 5: Make the pivot in the third column a `1` (if applicable).

In this case, the third row is all zeros, so no further operations are needed.

Final RREF Matrix:

$$\begin{bmatrix} 1 & 0 & -2 & -3 \\ 0 & 1 & 1 & 4 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

```python
from sympy import Matrix

# Define the augmented matrix
A = Matrix([
    [1, 3, 1, 9],
    [1, 1, -1, 1],
    [3, 11, 5, 35]
])

# Perform row reduction (Gauss-Jordan)
reduced_matrix = A.rref()

# Display the result
print("Reduced Row Echelon Form:")
print(reduced_matrix[0])
```

reduced_matrix = (M_rref, pivot_columns)

Matrix Rank

- **Definition**:
  The rank of a matrix A is the maximum number of linearly independent rows or columns in A.
- **Key Points**:
  - Rank represents the **dimension of the column space** or **row space** of the matrix.
  - A matrix is **full rank** if its rank equals the smaller of its dimensions ($\min(n, m)$ for an $n \times m$ matrix).
- **Properties**:
  - $\text{rank}(A) \leq \min(n, m)$.
  - Rank remains unchanged under row or column operations.
  - If A is square and full rank, $\det(A) \neq 0$.
- **Example**:
  For $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$,
  $\text{rank}(A) = 2$, as only 2 rows/columns are linearly independent.

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

Two **columns** are linearly independent

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

Two **rows** are linearly independent

Exercise

Determine whether the set of vectors $\{v_1, v_2, v_3\}$ is linearly independent or dependent over the field of real numbers ($\mathbb{R}$) in each case.

Part (a)

$$v_1 = \begin{bmatrix} 1 \\ -1 \\ 2 \end{bmatrix}, \quad v_2 = \begin{bmatrix} 1 \\ 1 \\ -2 \end{bmatrix}, \quad v_3 = \begin{bmatrix} -2 \\ 3 \\ 1 \end{bmatrix}$$

Part (b)

$$v_1 = \begin{bmatrix} 1 \\ 3 \\ 3 \end{bmatrix}, \quad v_2 = \begin{bmatrix} -1 \\ 1 \\ 3 \end{bmatrix}, \quad v_3 = \begin{bmatrix} -5 \\ -7 \\ 3 \end{bmatrix}$$

Part (a): Linear Independence or Dependence

To check if the vectors are linearly independent, solve the equation:

$$c_1 v_1 + c_2 v_2 + c_3 v_3 = 0$$

This translates to solving the following augmented system:

$$\begin{bmatrix} 1 & 1 & -2 & 0 \\ -1 & 1 & 3 & 0 \\ 2 & -2 & 1 & 0 \end{bmatrix}$$

Part (b): Linear Independence or Dependence

Similarly, solve the equation:

$$c_1 v_1 + c_2 v_2 + c_3 v_3 = 0$$

This translates to solving:

$$\begin{bmatrix} 1 & -1 & -5 & 0 \\ 3 & 1 & -7 & 0 \\ 3 & 3 & 3 & 0 \end{bmatrix}$$

linear_independence_check.py

```python
from sympy import Matrix

# Part (a): Define the matrix with vectors as columns
A = Matrix([
    [1, 1, -2],
    [-1, 1, 3],
    [2, -2, 1]
])

# Compute the rank of the matrix
rank_a = A.rank()

# Check linear independence for Part (a)
if rank_a == A.shape[1]:
    result_a = "Linearly Independent"
else:
    result_a = "Linearly Dependent"

# Part (b): Define the matrix with vectors as columns
B = Matrix([
    [1, -1, -5],
    [3, 1, -7],
    [3, 3, 3]
])

# Compute the rank of the matrix
rank_b = B.rank()

# Check linear independence for Part (b)
if rank_b == B.shape[1]:
    result_b = "Linearly Independent"
else:
    result_b = "Linearly Dependent"

# Print results
print("Part (a):", result_a)
print("Part (b):", result_b)
```

unitvector.py

Unit Vector

- **Definition**:

  A unit vector is a vector with **magnitude equal to 1**, also referred to as a direction vector.

- **Formula**:

  The unit vector $\hat{\mathbf{v}}$ in the direction of a given nonzero vector $\mathbf{v}$ is computed as:

$$\hat{\mathbf{v}} = \frac{\mathbf{v}}{\|\mathbf{v}\|}$$

  where $\|\mathbf{v}\|$ is the norm (magnitude) of $\mathbf{v}$.

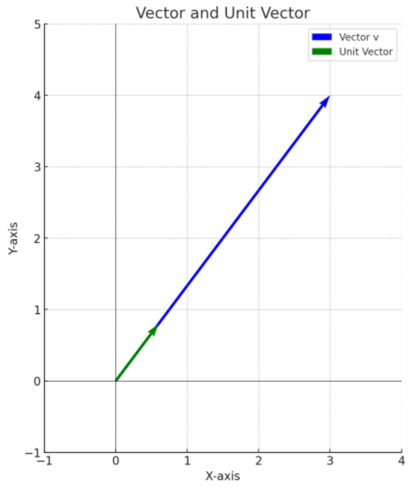
- **Properties**:

  - Unit vectors retain the **direction** of $\mathbf{v}$.
  - Any vector can be converted to a unit vector by dividing it by its norm.

- **Example**:

  For $\mathbf{v} = [3, 4]$,

$$\|\mathbf{v}\| = \sqrt{3^2 + 4^2} = 5, \quad \hat{\mathbf{v}} = \frac{[3, 4]}{5} = [0.6, 0.8].$$




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

# Define the vector v
v = np.array([3, 4])

# Compute the magnitude (norm) of v
norm_v = np.linalg.norm(v)

# Compute the unit vector of v
unit_v = v / norm_v

# Create a 2D plot
plt.figure()
plt.quiver(0, 0, v[0], v[1], angles='xy', scale_units='xy', scale=1, color='blue', label='Vector v')
plt.quiver(0, 0, unit_v[0], unit_v[1], angles='xy', scale_units='xy', scale=1, color='green', label='Unit Vector v̂')

# Set axis limits
plt.xlim(0, 4)
plt.ylim(0, 4.5)

# Add grid, labels, and legend
plt.grid()
plt.axhline(0, color='black', linewidth=0.5)
plt.axvline(0, color='black', linewidth=0.5)
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.title('Vector and Unit Vector')

# Show the plot
plt.show()
```

Linear Algebraic Equations

Definition:

Given the equation:

$$A\mathbf{x} = \mathbf{y},$$

where:

- A: an $m \times n$ real matrix,
- $\mathbf{x}$: an $n \times 1$ unknown vector,
- $\mathbf{y}$: an $m \times 1$ vector of constants.

The goal is to solve for $\mathbf{x}$, where:

- m: the number of equations,
- n: the number of unknowns.

Key Concepts:

1. **Range Space of A:**
   - The range space (or column space) of A consists of all possible linear combinations of the columns of A.
2. **Rank of A:**
   - Defined as the dimension of the range space (number of linearly independent columns of A).
3. **Null Vector and Null Space:**
   - A vector $\mathbf{x}$ is a null vector if $A\mathbf{x} = 0$.
   - The null space consists of all null vectors of A.
4. **Nullity of A:**
   - The nullity is the dimension of the null space, i.e., the number of linearly independent null vectors of A.
   - Related to the rank by:

$$\text{Nullity}(A) = \text{Number of Columns of } A - \text{Rank}(A).$$

Exercise

Consider the matrix:

$$A = \begin{bmatrix} 0 & 1 & 1 & 2 \\ 1 & 2 & 3 & 4 \\ 2 & 0 & 2 & 0 \end{bmatrix}$$

1. Determine the rank of A.
2. Identify the basis of the range space (column space) of A.
3. Compute the nullity of A.
4. Find the basis of the null space of A.

Finding the Basis:

1. **Null Space Definition**: The null space of a matrix A is the set of all vectors $\mathbf{n}$ such that $A\mathbf{n} = 0$.
2. **Solve $A\mathbf{n} = 0$**:
   - Write the augmented matrix for the homogeneous system $A\mathbf{n} = 0$.
   - Perform row reduction to bring the matrix to its reduced row echelon form (RREF).
   - Express the free variables in terms of the pivot variables to get the general solution.
3. **Identify Basis Vectors**:
   - The solution vector will include parameters (free variables).
   - Each parameter corresponds to a linearly independent vector in the null space.
   - Collect these vectors to form the basis of the null space.

Solution

1. **Rank of A:**
   - A has 2 linearly independent columns ($\mathbf{a}_1$ and $\mathbf{a}_2$).
   - Rank(A) = 2.
2. **Basis of the Range Space:**
   - The basis of the column space is formed by the first two columns:

$$\text{Basis of Range Space: } \{\mathbf{a}_1, \mathbf{a}_2\} = \left\{ \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}, \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix} \right\}$$

3. **Nullity of A:**
   - Using the formula:

$$\text{Nullity}(A) = \text{Number of Columns} - \text{Rank}(A) = 4 - 2 = 2$$

4. **Basis of the Null Space:**
   - Solve $A\mathbf{n} = 0$:

$$\mathbf{n}_1 = \begin{bmatrix} 1 \\ 1 \\ -1 \\ 0 \end{bmatrix}, \quad \mathbf{n}_2 = \begin{bmatrix} 0 \\ 2 \\ 0 \\ -1 \end{bmatrix}$$

   - The basis of the null space is:

$$\text{Basis of Null Space: } \{\mathbf{n}_1, \mathbf{n}_2\}$$

Transpose of a Matrix

Transpose of a Matrix

- **Definition**:
  The transpose of a matrix A, denoted as A^T, is a new matrix formed by interchanging the rows and columns of A.
- **Notation**:
  If $A = [a_{ij}]$, then $A^T = [a_{ji}]$.
- **Properties**:
  1. $(A^T)^T = A$
  2. $(A+B)^T = A^T + B^T$
  3. $(cA)^T = cA^T$ (for scalar c)
  4. $(AB)^T = B^T A^T$
- **Example**:
  If

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix},$$

then

$$A^T = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}.$$

```python
import numpy as np

# Define the matrix
A = np.array([
    [1, 2],
    [3, 4],
    [5, 6]
])

# Compute the transpose
A_transpose = A.T

# Display the original matrix and its transpose
print("Original Matrix A:")
print(A)

print("\nTranspose of Matrix A (A^T):")
print(A_transpose)
```

Eigenvalues and Eigenvectors

Eigenvalues and Eigenvectors

- **Definition**:
  For a square matrix A, an eigenvector $\mathbf{v}$ and eigenvalue λ satisfy:

$$A\mathbf{v} = \lambda\mathbf{v},$$

  where $\mathbf{v} \neq 0$ is the eigenvector and λ is the eigenvalue.

- **Properties**:
  1. Eigenvalues can be real or complex.
  2. The eigenvectors corresponding to distinct eigenvalues are linearly independent.
  3. The determinant of $A - \lambda I = 0$ gives the eigenvalues.

- **Applications**:
  Used in stability analysis, dimensionality reduction (PCA), vibration analysis, etc.

- **Example**:
  For matrix

$$A = \begin{bmatrix} 4 & 1 \\ 2 & 3 \end{bmatrix},$$

  - Eigenvalues: $\lambda_1 = 5, \lambda_2 = 2$
  - Eigenvectors:
    $\mathbf{v}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \mathbf{v}_2 = \begin{bmatrix} -1 \\ 2 \end{bmatrix}.$

```python
import numpy as np

# Define the matrix
A = np.array([
    [4, 1],
    [2, 3]
])

# Compute eigenvalues and eigenvectors
eigenvalues, eigenvectors = np.linalg.eig(A)

# Display the results
print("Matrix A:")
print(A)

print("\nEigenvalues of A:")
print(eigenvalues)

print("\nEigenvectors of A (columns correspond to eigenvalues):")
print(eigenvectors)
```

1. Scaling of Eigenvectors

- Eigenvectors associated with an eigenvalue are unique only up to a **scaling factor** (multiplication by a nonzero constant).
- If $\mathbf{v}$ is an eigenvector, then any scaled version $c\mathbf{v}$ (where $c \neq 0$) is also a valid eigenvector.

Example:
If an eigenvector is $[1, 1]^T$, another valid one could be $[0.707, 0.707]^T$ (normalized) or $[2, 2]^T$ (scaled).

Exercise: Stability Analysis of a Linear System with Varying Dynamics

We aim to analyze the behavior of a linear system of first-order differential equations under different dynamic conditions, characterized by different choices of the system matrix A. The system is given by:

$$\dot{\mathbf{x}} = A\mathbf{x}$$

where:

$$A \in \mathbb{R}^{2\times 2}, \quad \mathbf{x}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix}.$$

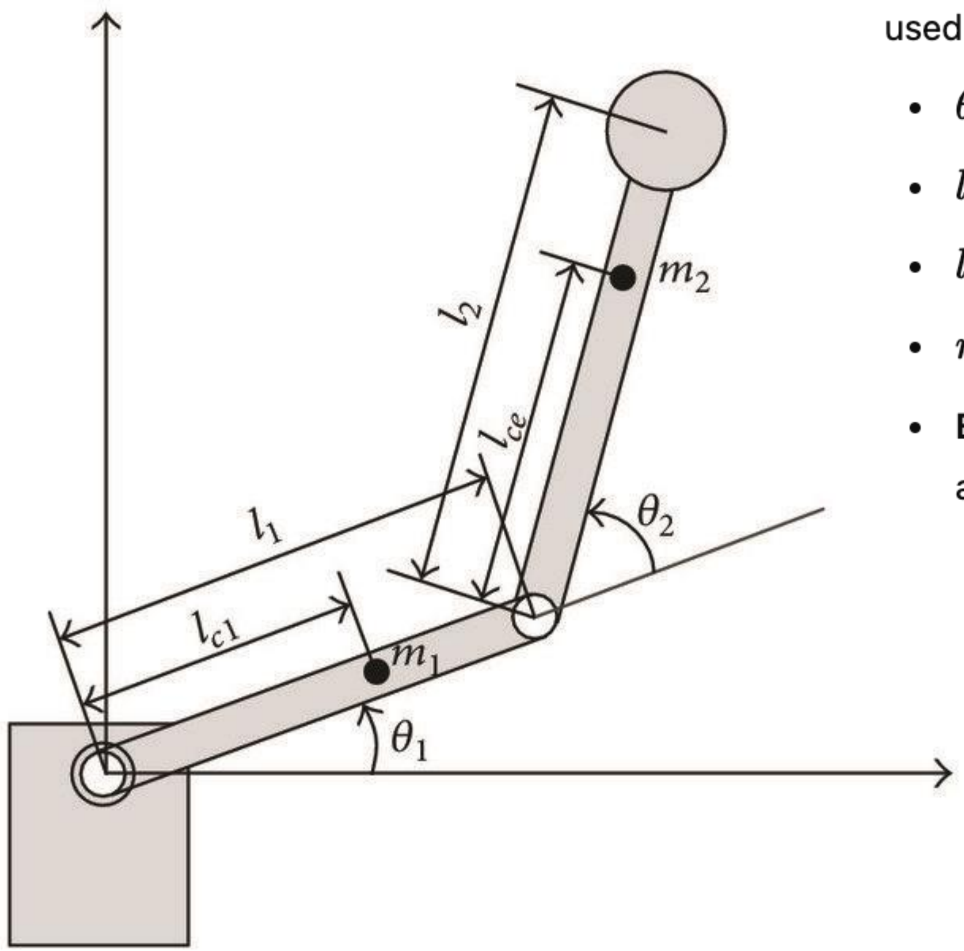
Instructions

1. **Define the system function**: Implement the differential equation in a Python function.
2. **Experiment with different system matrices**: Choose different matrices A that yield:
   - Real and distinct eigenvalues.
   - Complex conjugate eigenvalues.
   - Repeated eigenvalues.
   - Eigenvalues with positive, negative, or zero real parts.
3. **Solve the system**: Use `solve_ivp` to integrate the differential equation over the time interval $t \in [0, 10]$.
4. **Explore different initial conditions**: Try multiple initial conditions and analyze how the system evolves in each case.
5. **Plot the solutions**: For each combination of A and initial conditions, plot $x_1(t)$ and $x_2(t)$

eigSystem.py

Extra point Robotics = Jose Herrera and Alan Lamoglia

Two-Link Planar Manipulator



The figure illustrates a **two-link planar robotic manipulator**, a common model in robotics used for kinematic and dynamic analysis. The notation follows standard conventions:

- θ_1, θ_2: Joint angles that define the position of each link relative to the previous one.
- l_1, l_2: Lengths of the first and second links, respectively.
- l_{c1}, l_{c2}: Distances from each joint to the center of mass of the respective links.
- m_1, m_2: Masses of the links, with their centers of mass indicated by the black dots.
- **End-effector (circular shape):** The terminal point of the second link, representing where a tool or sensor would be attached.

Exercise: Singularity Analysis of a 2-DOF Robotic Manipulator

Problem Statement

A **2-DOF planar robotic manipulator** consists of two links with lengths L_1 and L_2, connected by two revolute joints. The end-effector position is determined by the joint angles θ_1 and θ_2.

In this exercise, you will **analyze the singularity conditions** of this manipulator by evaluating its **Jacobian matrix**.

Given Jacobian Matrix

The Jacobian matrix $J(\theta_1, \theta_2)$ for the manipulator's end-effector velocity is:

$$J(\theta_1, \theta_2) = \begin{bmatrix} -L_1 \sin\theta_1 - L_2 \sin(\theta_1 + \theta_2) & -L_2 \sin(\theta_1 + \theta_2) \\ L_1 \cos\theta_1 + L_2 \cos(\theta_1 + \theta_2) & L_2 \cos(\theta_1 + \theta_2) \end{bmatrix}$$

Tasks

1. **Compute the determinant of** J and determine when $\det(J) = 0$.
2. **Find the singular configurations**, i.e., values of θ_1 and θ_2 where the manipulator loses rank.
3. **Explain the effect of singularities** on the robot's motion. What happens when the determinant is zero?
4. **Visualize the singular configurations** by plotting them over a range of joint values.

💡 **Hint**: A singularity occurs when the determinant of the Jacobian is zero, meaning the robot loses the ability to move in certain directions.

What is the Jacobian Matrix? 📌

The **Jacobian matrix** in robotics describes how the motion of a robot's **joint variables** (e.g., angles of revolute joints) translates into the **velocity of the end-effector** in Cartesian space.

For a **robotic manipulator**, the Jacobian J is a matrix that relates the joint velocities $\dot{\theta}$ to the **linear** and **angular velocity** of the end-effector:

$$\dot{\mathbf{x}} = J(\boldsymbol{\theta})\dot{\boldsymbol{\theta}}$$

where:

- $\dot{\mathbf{x}}$ represents the end-effector velocity,
- $J(\boldsymbol{\theta})$ is the Jacobian matrix, which depends on the current joint angles,
- $\dot{\boldsymbol{\theta}}$ represents the joint velocities.

Why is the Jacobian Important?

- It determines how joint movements affect the robot's position and orientation.
- It helps in **inverse kinematics**, where we compute the required joint velocities to achieve a desired motion.
- It is crucial for **singularity analysis**, since when $\det(J) = 0$, the robot loses mobility in some directions, making it harder or impossible to move in certain ways.

👉 **In this exercise, we will analyze when the Jacobian becomes singular and what that means for the manipulator's motion.**

Solution:

Let's simplify the determinant expression step by step.

We start with:

$$\det(J) = (-L_1 \sin\theta_1 - L_2 \sin(\theta_1+\theta_2))(L_2 \cos(\theta_1+\theta_2)) - (-L_2 \sin(\theta_1+\theta_2))(L_1 \cos\theta_1 + L_2 \cos(\theta_1+\theta_2))$$

Step 1: Expand Both Terms

Expanding the first product:

$$(-L_1 \sin\theta_1 - L_2 \sin(\theta_1+\theta_2)) \cdot L_2 \cos(\theta_1+\theta_2)$$

$$= -L_1 L_2 \sin\theta_1 \cos(\theta_1+\theta_2) - L_2^2 \sin(\theta_1+\theta_2)\cos(\theta_1+\theta_2)$$

Expanding the second product:

$$(-L_2 \sin(\theta_1+\theta_2)) \cdot (L_1 \cos\theta_1 + L_2 \cos(\theta_1+\theta_2))$$

$$= -L_1 L_2 \sin(\theta_1+\theta_2)\cos\theta_1 - L_2^2 \sin(\theta_1+\theta_2)\cos(\theta_1+\theta_2)$$

Step 2: Combine Like Terms

$$\det(J) = (-L_1 L_2 \sin\theta_1 \cos(\theta_1+\theta_2) - L_2^2 \sin(\theta_1+\theta_2)\cos(\theta_1+\theta_2))$$

$$-(-L_1 L_2 \sin(\theta_1+\theta_2)\cos\theta_1 - L_2^2 \sin(\theta_1+\theta_2)\cos(\theta_1+\theta_2))$$

Distribute the negative sign:

$$= -L_1 L_2 \sin\theta_1 \cos(\theta_1+\theta_2) - L_2^2 \sin(\theta_1+\theta_2)\cos(\theta_1+\theta_2)$$

$$+L_1 L_2 \sin(\theta_1+\theta_2)\cos\theta_1 + L_2^2 \sin(\theta_1+\theta_2)\cos(\theta_1+\theta_2)$$

Cancel the common terms:

$$\det(J) = -L_1 L_2 \sin\theta_1 \cos(\theta_1+\theta_2) + L_1 L_2 \sin(\theta_1+\theta_2)\cos\theta_1$$

Step 3: Factor Terms

Factor $L_1 L_2$:

$$\det(J) = L_1 L_2 (\sin(\theta_1 + \theta_2) \cos\theta_1 - \sin\theta_1 \cos(\theta_1 + \theta_2))$$

Using the **sine angle subtraction identity**:

$$\sin A \cos B - \sin B \cos A = \sin(A - B)$$

with $A = \theta_1 + \theta_2$ and $B = \theta_1$:

$$\sin(\theta_1 + \theta_2) \cos\theta_1 - \sin\theta_1 \cos(\theta_1 + \theta_2) = \sin(\theta_1 + \theta_2 - \theta_1) = \sin\theta_2$$

Thus, the determinant simplifies to:

$$\det(J) = L_1 L_2 \sin\theta_2$$

Final Result

$$\det(J) = L_1 L_2 \sin\theta_2$$

Singular Configurations

The determinant is zero when:

$$L_1 L_2 \sin\theta_2 = 0$$

Since L_1 and L_2 are nonzero, we get:

$$\sin\theta_2 = 0 \quad \Rightarrow \quad \theta_2 = 0, \pi, 2\pi, \ldots$$

This means the **robot is in a singular configuration when $\theta_2 = 0$ or $\theta_2 = \pi$**, which corresponds to when the manipulator is fully extended or fully folded.