# NumPy in Python

The NumPy library in Python is widely used for linear algebra due to its powerful array handling capabilities. It provides efficient functions for a range of linear algebra operations, such as matrix creation, matrix operations, determinants, eigenvalues, and linear equation solving.

- Python Array Dimensions: Understanding Shape and Size
- Reshape an Array
- Python NumPy Tutorial: Indexing and Slicing
- Vertically and Horizontally Stack Arrays with vstack and hstack
- How to Find Unique Values in an Array
- How to Handle Missing Values in NumPy with Masked Arrays
- Generate a Random Number Between 0 and 1 in Python with #NumPy
- 3 Ways to Use a Random Generator

# Linear Algebra with NumPy

# 1. Creating Matrices and Arrays

The np.array() function is used to create matrices (2D arrays) and vectors (1D arrays).

```
import numpy as np
# Creating a vector
vector = np.array([1, 2, 3])
# Creating a 2x2 matrix
matrix = np.array([[1, 2], [3, 4]])
# Creating a 3x3 matrix with zeros
zero_matrix = np.zeros((3, 3))
# Creating a 3x3 identity matrix
identity_matrix = np.eye(3)
```

- How to Create 1D, 2D, and 3D Arrays in NumPy
- Python NumPy Array Creation Methods Explained | Zeros, Ones, Empty, Arange, and Linspace

### 2. Basic Matrix Operations

Basic arithmetic operations are easy with NumPy arrays and can be performed element-wise or with specific matrix functions.

- Addition and Subtraction are element-wise.
- Scalar Multiplication multiplies each element by a scalar value.
- Matrix Multiplication (dot product) can be done with np.dot() or the @ operator.

```
A = np.array([[1, 2], [3, 4]])
B = np.array([[2, 0], [1, 3]])
# Element-wise addition and subtraction
addition = A + B
subtraction = A - B
# Scalar multiplication
scalar_multiplication = 2 * A
# Matrix multiplication
matrix_multiplication = np.dot(A, B) # or A @ B
```

- Mastering Arithmetic and Logical Operations for Efficient Data Processing
- Matrix Multiplication using NumPy in Python

#### 3. Transpose of a Matrix

Transposing a matrix (switching rows and columns) is straightforward with the .T attribute.

A = np.array([[1, 2], [3, 4]]) transpose\_A = A.T

Matrix Transpose using NumPy in Python

### 4. Determinants

The determinant is a scalar value that can be computed using np.linalg.det(), which is useful in solving linear systems and understanding matrix properties.

A = np.array([[1, 2], [3, 4]])
det\_A = np.linalg.det(A) # Output: -2.0

### 5. Inverse of a Matrix

The inverse of a matrix (A) is another matrix (A $^{-1}$ ) such that (A \times A $^{-1}$  = I) (the identity matrix). Use np.linalg.inv() to find the inverse.

```
A = np.array([[1, 2], [3, 4]])
inverse_A = np.linalg.inv(A)
```

• Inverse of a Matrix in Python with NumPy

### 6. Solving Systems of Linear Equations

For a system of equations of the form (Ax = B), where (A) is a matrix of coefficients, (x) is a vector of unknowns, and (B) is a vector of constants, we can solve for (x) using np.linalg.solve().

```
A = np.array([[3, 1], [1, 2]])
B = np.array([9, 8])
solution = np.linalg.solve(A, B)
```

### 7. Eigenvalues and Eigenvectors

Eigenvalues and eigenvectors are fundamental in linear transformations, physics, and machine learning. Use np.linalg.eig() to find them.

```
A = np.array([[4, -2], [1, 1]])
eigenvalues, eigenvectors = np.linalg.eig(A)
```

# 8. Norms of Vectors and Matrices

The norm of a vector or matrix gives a measure of its magnitude. np.linalg.norm() calculates the norm, and you can specify the order (e.g., 1-norm, 2-norm, or infinity norm).

```
vector = np.array([3, 4])
matrix = np.array([[1, 2], [3, 4]])
# Euclidean norm (default is 2-norm)
norm_vector = np.linalg.norm(vector)
# Frobenius norm (for matrices)
norm_matrix = np.linalg.norm(matrix, 'fro')
```

# 9. Rank of a Matrix

The rank of a matrix represents the maximum number of linearly independent row or column vectors. np.linalg.matrix\_rank() provides the rank of a matrix.

A = np.array([[1, 2], [2, 4]])
rank\_A = np.linalg.matrix\_rank(A)

# 10. Singular Value Decomposition (SVD)

SVD decomposes a matrix into three matrices, useful in applications like image compression and data reduction. np.linalg.svd() provides this decomposition.

```
A = np.array([[1, 2], [3, 4], [5, 6]])
U, S, V = np.linalg.svd(A)
```

### Summary Table

Operation	Function or Method	Description
Matrix Creation	np.array(),np.zeros(), np.eye()	Create matrices and arrays
Addition, Subtraction	A + B, A - B	Element-wise addition and subtraction
Scalar Multiplication	2 * A	Multiply each element by a scalar
Matrix Multiplication	np.dot(A, B) or A @ B	Standard matrix multiplication
Transpose	A.T	Transpose of a matrix
Determinant	np.linalg.det(A)	Scalar value representing matrix characteristics
Inverse	<pre>np.linalg.inv(A)</pre>	Inverse of a matrix
Solve Linear Equations	<pre>np.linalg.solve(A, B)</pre>	Solves ( $Ax = B$ )
Eigenvalues and Eigenvectors	np.linalg.eig(A)	Returns eigenvalues and eigenvectors
Norms	<pre>np.linalg.norm()</pre>	Magnitude of vector/matrix
Rank	<pre>np.linalg.matrix_rank(A)</pre>	Max linearly independent vectors
Singular Value Decomposition	np.linalg.svd(A)	Decomposes matrix into U, S, V components

NumPy is highly optimized for numerical calculations, making it ideal for complex linear algebra applications. Let me know if you'd like further explanations or examples for any of these functions!

# Key Terms

True/False (Mark T for True and F for False)

Answer Key (True/False):

Multiple Choice (Select the best answer)

### 1. Which function would you use to determine the type of a variable in Python?

- A) id()
- B) type()

- C) str()
- D) isinstance()

Watch this video for the answer:

Answer key (Mutiple Choice):

# Fill in the Blanks

Answer Key (Fill in the Blanks):

# Exercises

Beginner: Basic concepts and syntax.

Intermediate: More complex problems involving data structures and algorithms.

#### 1. Exercise: Access Reverse Diagonal Elements in a 2D List Based on Index Sum

**Problem Statement:** Write a Python program to access elements from a 2D list based on a specific condition. The 2D list represents a matrix of integers, and your task is to use nested loops to find and print the elements where the sum of the row index (i) and the column index (j) equals 2. These elements form a reverse diagonal in the matrix.

#### Input:

You are given a 3x3 matrix arr:

arr = [ [0, 1, 2], # Row 0 [3, 4, 5], # Row 1 [6, 7, 8] # Row 2 ]

#### **Output:**

Print the elements of the matrix where the sum of the row index and column index equals 2, along with their positions.

#### **Example Output:**

Element at (0,2) is 2 Element at (1,1) is 4 Element at (2,0) is 6

Advanced: Challenging problems that require in-depth understanding and optimization.

# **Review Questions**

# **References and Bibliography**

For more details, see Appendix A.

# **Appendices**

# **Appendix A: Data Tables**

This section includes the data tables referred to in the text...