Assignment III: Linear Algebra and Matrix Theory

Submitted To

Samriddha Pathak

Digital Pathshala Instructor

Submitted By

Rajendra Chimala

Submitted At

2025-July-10

Contents

QN-1. Define a linear transformation. Show whether the function $T(x, y) = (2x, 3y)$ is a linear transformation by checking the two required properties.

In linear algebra the linear transformation is the function that holds the property of additivity and homogeneity for all vector and scaler is called linear transformation. That additivity property is $T(u+v) = T(u) + T(v)$ and the homogeneity property is $T(cu) = cT(u)$.

Here, we have given: $T(x,y) = (2x,3y)$

Let assume $u = (x_1, y_1)$ and $v = (x_2,y_2)$

Now we need to check the both properties to know the It is linear transformation or not:

Let's check the additive property:

$T(u+v) = T(x_1 + x_2, y_1 + y_2)$

$= (2(x_1 + x_2), 3(y_1 + y_2))$

$= (2x_1 + 2x_2, 3y_1 + 3y_2)$

$T(u) + T(v) = (2x_1, 3y_1) + (2x_2 + 3y_2)$

$= (2x_1 + 2x_2, 3y_1 + 3y_2)$

: It holds the Additive property.

Let's check the homogeneity property

$T(cu) = cT(u)$

$T(cu) = T((cx_1, cy_1))$

$= (2(cx_1), 3(cy_1))$

$= (c\cdot 2x_1, c\cdot 3y_1)$

$cT(u) = c\cdot(2x_1,3y_1)$

$= (c\cdot 2x_1, c\cdot 3y_1)$

:**It holds homogeneity property.**

: **The given function T(x,y)=(2x,3y) T(x, y) = (2x, 3y) T(x,y)=(2x,3y) is a linear transformation because it holds the both properties.**

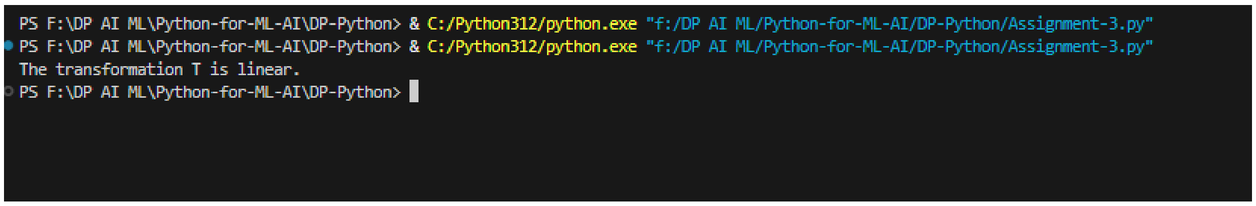
QN2: Write a NumPy function that checks if a given transformation matrix satisfies additivity and scalar multiplication preservation (i.e., linearity).

Here the code for check the transformation matrix satisfies additivity and scaler multiplication:

Click here to check code on online compiler

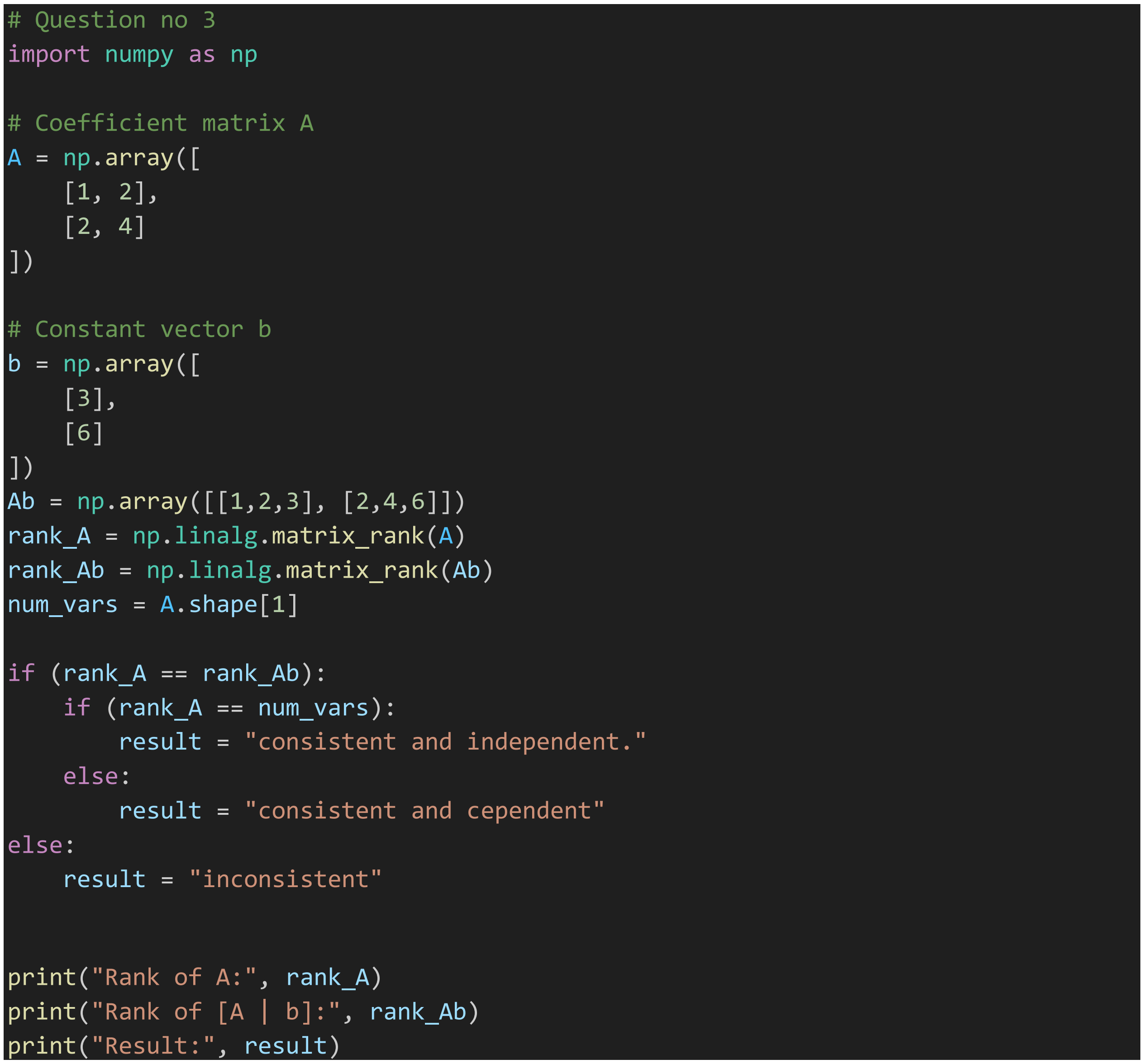
```python
# Assignment 3 Solution
import numpy as np

def is_linear_transformation(T_matrix, u, v, scalar, tol=1e-8):


    def T(x):
        return T_matrix @ x

    additivity = np.allclose(T(u + v), T(u) + T(v), atol=tol)

    homogeneity = np.allclose(T(scalar * u), scalar * T(u), atol=tol)

    return additivity and homogeneity

def main():

    T_matrix = np.array([[2, 0], [0, 3]])

    u = np.array([1, 2])
    v = np.array([3, 4])

    scalar = 5

    result = is_linear_transformation(T_matrix, u, v, scalar)
    if (result==True):
        print("The transformation T is linear.")
    else:
        print("The transformation T is not linear.")


main()
```

Output:



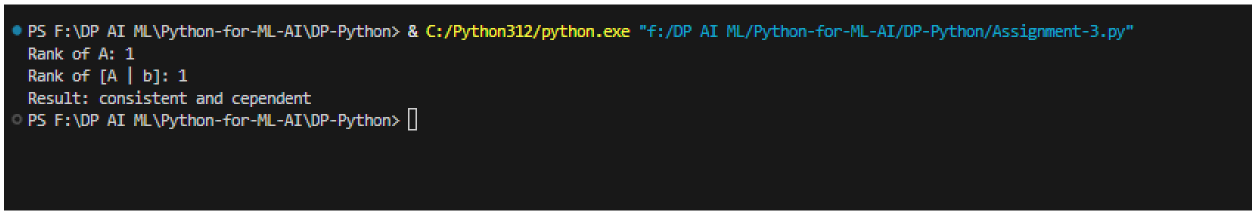
```
PS F:\DP AI ML\Python-for-ML-AI\DP-Python> & C:/Python312/python.exe "f:/DP AI ML/Python-for-ML-AI/DP-Python/Assignment-3.py"
PS F:\DP AI ML\Python-for-ML-AI\DP-Python> & C:/Python312/python.exe "f:/DP AI ML/Python-for-ML-AI/DP-Python/Assignment-3.py"
The transformation T is linear.
PS F:\DP AI ML\Python-for-ML-AI\DP-Python>
```

QN3 Classify the following system as consistent/inconsistent and dependent /independent: $x + 2y = 3$, $2x + 4y = 6$ Justify your answer using matrix rank



```python
# Question no 3
import numpy as np

# Coefficient matrix A
A = np.array([
    [1, 2],
    [2, 4]
])

# Constant vector b
b = np.array([
    [3],
    [6]
])
Ab = np.array([[1,2,3], [2,4,6]])
rank_A = np.linalg.matrix_rank(A)
rank_Ab = np.linalg.matrix_rank(Ab)
num_vars = A.shape[1]

if (rank_A == rank_Ab):
    if (rank_A == num_vars):
        result = "consistent and independent."
    else:
        result = "consistent and cependent"
else:
    result = "inconsistent"


print("Rank of A:", rank_A)
print("Rank of [A | b]:", rank_Ab)
print("Result:", result)
```

Output



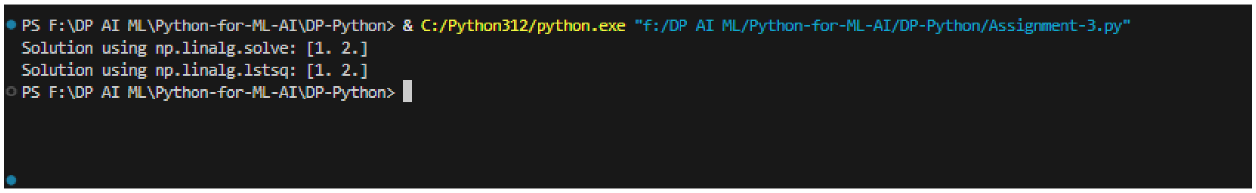
```
PS F:\DP AI ML\Python-for-ML-AI\DP-Python> & C:/Python312/python.exe "f:/DP AI ML/Python-for-ML-AI/DP-Python/Assignment-3.py"
Rank of A: 1
Rank of [A | b]: 1
Result: consistent and cependent
PS F:\DP AI ML\Python-for-ML-AI\DP-Python>
```

QN4. Solve the system of equations using both NumPy's np.linalg.solve and np.linalg.lstsq. Compare the results: 2x + 3y = 8, 5x + 4y = 13

Here are the solution for the given equation is below using the numpy:

```
# Question no 4
import numpy as np


A = np.array([[2, 3],
              [5, 4]])
b = np.array([8, 13])

# By using np.linalg.solve
x_solve = np.linalg.solve(A, b)

# By using np.linalg.lstsq
x_lstsq = np.linalg.lstsq(A, b, rcond=None)[0]

print("Solution using np.linalg.solve:", x_solve)
print("Solution using np.linalg.lstsq:", x_lstsq)
```

Output:



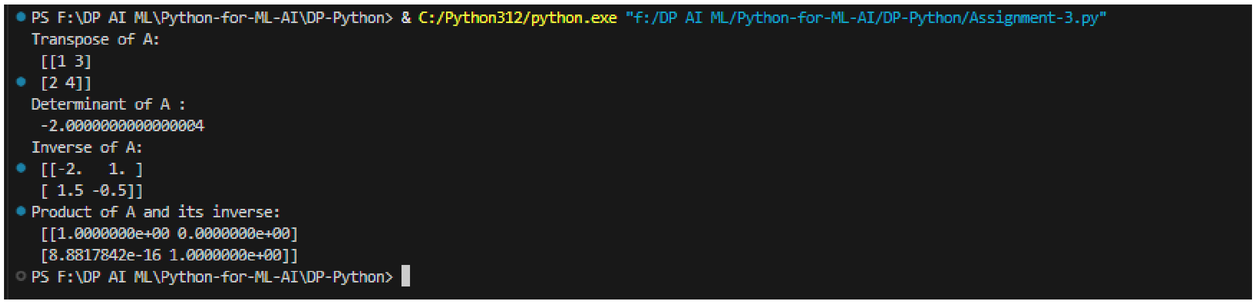
```
PS F:\DP AI ML\Python-for-ML-AI\DP-Python> & C:/Python312/python.exe "f:/DP AI ML/Python-for-ML-AI/DP-Python/Assignment-3.py"
Solution using np.linalg.solve: [1. 2.]
Solution using np.linalg.lstsq: [1. 2.]
PS F:\DP AI ML\Python-for-ML-AI\DP-Python>
```

QN5. Given the matrix A = [[1, 2] [3, 4]], compute the following using NumPy: # Transpose # Inverse # Determinant AA^{-1} = I

```
# QN 5
import numpy as np
```

```python
A = np.array([[1, 2], [3, 4]])
# Transpose of A
transposed = A.T
print("Transpose of A:\n", transposed)

# Determinant of A
determinant = np.linalg.det(A)
print("Determinant of A : \n", determinant)

# Inverse of A
inverse = np.linalg.inv(A)
print("Inverse of A:\n", inverse)

# AA-1
res = A@inverse
print("Product of A and its inverse:\n", res)
```

Output:

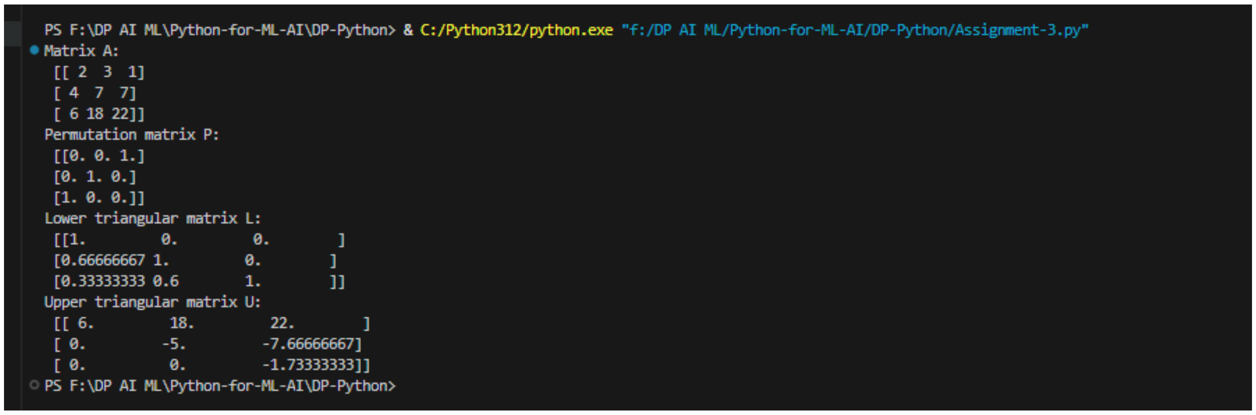


```
PS F:\DP AI ML\Python-for-ML-AI\DP-Python> & C:/Python312/python.exe "f:/DP AI ML/Python-for-ML-AI/DP-Python/Assignment-3.py"
Transpose of A:
 [[1 3]
 [2 4]]
Determinant of A :
 -2.0000000000000004
Inverse of A:
 [[-2.   1. ]
 [ 1.5 -0.5]]
Product of A and its inverse:
 [[1.0000000e+00 0.0000000e+00]
 [8.8817842e-16 1.0000000e+00]]
PS F:\DP AI ML\Python-for-ML-AI\DP-Python>
```

6. Perform LU decomposition on a 3×3 matrix using SciPy. Interpret the resulting matrices L, U, P and describe their utility in solving linear systems

```python
# QN 6

import numpy as np
from scipy.linalg import lu

A = np.array([[2, 3, 1],
              [4, 7, 7],
              [6, 18, 22]])

P, L, U = lu(A)

print("Matrix A:\n", A)
print("Permutation matrix P:\n", P)
print("Lower triangular matrix L:\n", L)
print("Upper triangular matrix U:\n", U)
```

Output:

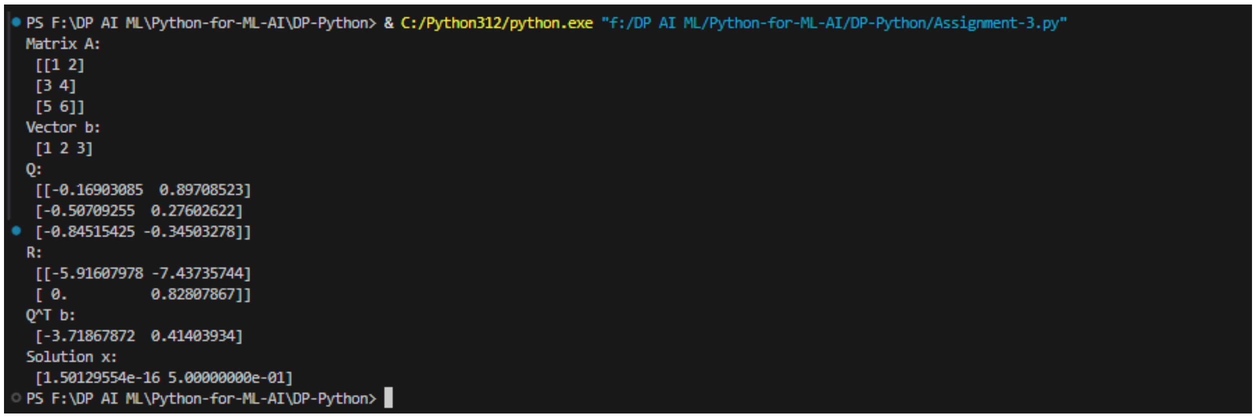


```
PS F:\DP AI ML\Python-for-ML-AI\DP-Python> & C:/Python312/python.exe "f:/DP AI ML/Python-for-ML-AI/DP-Python/Assignment-3.py"
Matrix A:
 [[ 2  3  1]
 [ 4  7  7]
 [ 6 18 22]]
Permutation matrix P:
 [[0. 0. 1.]
 [0. 1. 0.]
 [1. 0. 0.]]
Lower triangular matrix L:
 [[1.         0.         0.        ]
 [0.66666667 1.         0.        ]
 [0.33333333 0.6        1.        ]]
Upper triangular matrix U:
 [[ 6.         18.         22.        ]
 [ 0.         -5.         -7.66666667]
 [ 0.          0.         -1.73333333]]
PS F:\DP AI ML\Python-for-ML-AI\DP-Python>
```

7. Solve the following overdetermined system using QR decomposition: A = [1,2],[3,4],[5,6] b=[1,2,3]

```python
# Question no 7
import numpy as np
from scipy.linalg import qr


A = np.array([[1, 2],
              [3, 4],
              [5, 6]])
b = np.array([1, 2, 3])


Q, R = qr(A, mode='economic')

Qt_b = np.dot(Q.T, b)

x = np.linalg.solve(R, Qt_b)

print("Matrix A:\n", A)
print("Vector b:\n", b)
print("Q:\n", Q)
print("R:\n", R)
print("Q^T b:\n", Qt_b)
print("Solution x:\n", x)
```

Output:



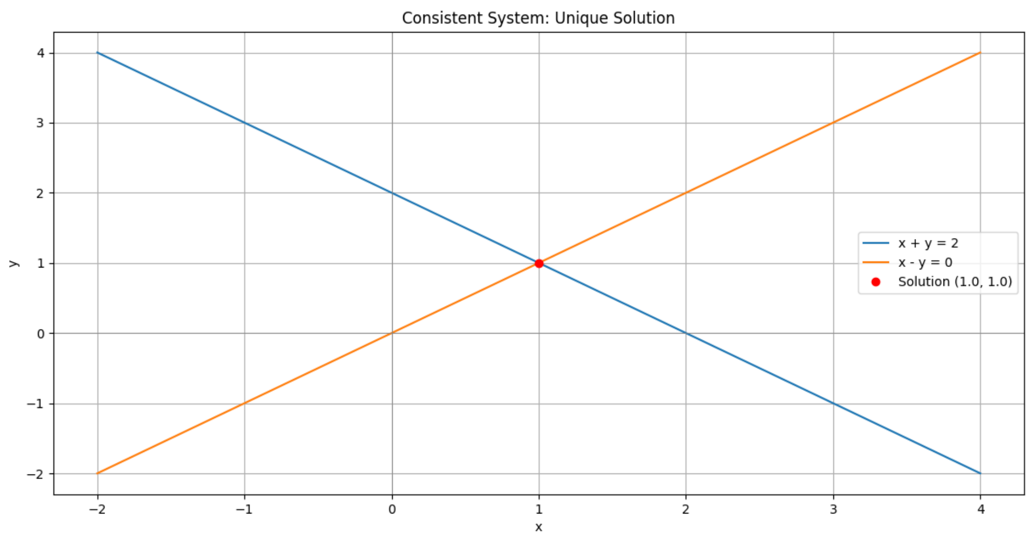
```
PS F:\DP AI ML\Python-for-ML-AI\DP-Python> & C:/Python312/python.exe "f:/DP AI ML/Python-for-ML-AI/DP-Python/Assignment-3.py"
Matrix A:
 [[1 2]
 [3 4]
 [5 6]]
Vector b:
 [1 2 3]
Q:
 [[-0.16903085  0.89708523]
 [-0.50709255  0.27602622]
 [-0.84515425 -0.34503278]]
R:
 [[-5.91607978 -7.43735744]
 [ 0.          0.82807867]]
Q^T b:
 [-3.71867872  0.41403934]
Solution x:
 [1.50129554e-16 5.00000000e-01]
PS F:\DP AI ML\Python-for-ML-AI\DP-Python>
```

8. Explain the geometric interpretation of consistent and inconsistent linear systems. Create and solve one example of each using NumPy, then visualize in 2D using matplotlib.

In two dimensions, every linear equation represents a straight line on the plane. To solve a system of two linear equations means to find the points of intersection of those lines. If the equations have at least one solution (for example if the lines intersect at some point or if they lie exactly on top of each other), then we say that the corresponding linear system is consistent; otherwise, for example if there is no solution (for example if the lines are parallel but with different intercepts), then we say that it is inconsistent.

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

A1 = np.array([[1, 1],
               [1, -1]])
b1 = np.array([2, 0])

x1 = np.linalg.solve(A1, b1)

x_vals = np.linspace(-2, 4, 100)
y1 = 2 - x_vals
y2 = x_vals

plt.figure(figsize=(6, 5))
plt.plot(x_vals, y1, label='x + y = 2')
plt.plot(x_vals, y2, label='x - y = 0')
plt.plot(x1[0], x1[1], 'ro', label=f'Solution ({x1[0]}, {x1[1]})')
plt.title('Consistent System: Unique Solution')
plt.xlabel('x'); plt.ylabel('y')
plt.axhline(0, color='gray', linewidth=0.5)
plt.axvline(0, color='gray', linewidth=0.5)
plt.legend()
plt.grid(True)
plt.show()
```

Output



9. Why is matrix invertibility important in solving linear systems? Give an example of a non-invertible matrix and interpret the result in terms of system solutions.

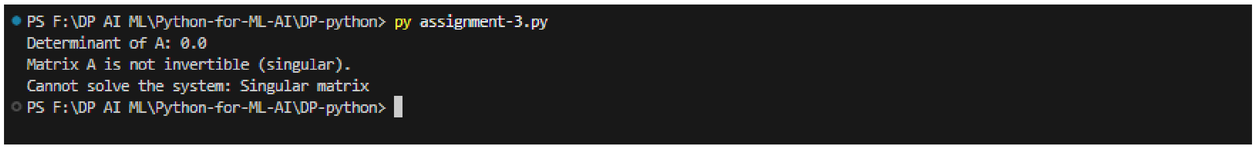
Matrix invertibility is useful in methods when working with systems of equations since it demonstrates if a linear system is solvable uniquely. When a system is written as Ax = b, and A is invertible, then x can be solved directly using.

$x=A^{-1}b$

```
import numpy as np

A = np.array([[2, 4],
              [1, 2]])

b = np.array([6, 3])

det_A = np.linalg.det(A)
print("Determinant of A:", det_A)
```

```python
if det_A == 0:
    print("Matrix A is not invertible (singular).")
else:
    x = np.linalg.inv(A) @ b
    print("Solution:", x)

try:
    x = np.linalg.solve(A, b)
    print("Solution:", x)
except np.linalg.LinAlgError as e:
    print("Cannot solve the system:", e)
```

Output:

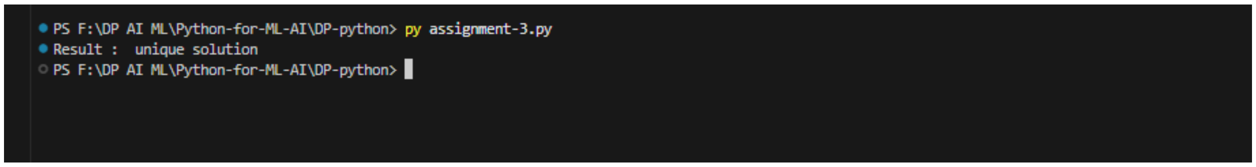


```
PS F:\DP AI ML\Python-for-ML-AI\DP-python> py assignment-3.py
Determinant of A: 0.0
Matrix A is not invertible (singular).
Cannot solve the system: Singular matrix
PS F:\DP AI ML\Python-for-ML-AI\DP-python>
```

10. Write a Python script using NumPy that classifies a given system AX = b as: (a) Consistent with a unique solution (b) Consistent with infinite solutions (b) Inconsistent

```python
import numpy as np

def classify_linear_system(A, b):
    A = np.array(A, dtype=float)
    b = np.array(b, dtype=float).reshape(-1, 1)

    augmented = np.hstack((A, b))

    rank_A = np.linalg.matrix_rank(A)
    rank_aug = np.linalg.matrix_rank(augmented)
    num_vars = A.shape[1]

    if rank_A == rank_aug:
        if rank_A == num_vars:
            return "unique solution"
        else:
            return "infinite solutions"
    else:
        return "Inconsistent."

A1 = [[2, 1],
      [1, -1]]
b1 = [4, 1]



print("Result : ", classify_linear_system(A1, b1))
```

Output:



```
PS F:\DP AI ML\Python-for-ML-AI\DP-python> py assignment-3.py
Result :  unique solution
PS F:\DP AI ML\Python-for-ML-AI\DP-python> ▌
```

11. Explain the difference between “basis of a vector space” and “basis of a column space” with a concrete example.

Here are the difference between the basis of a vector space and basis of a column space:

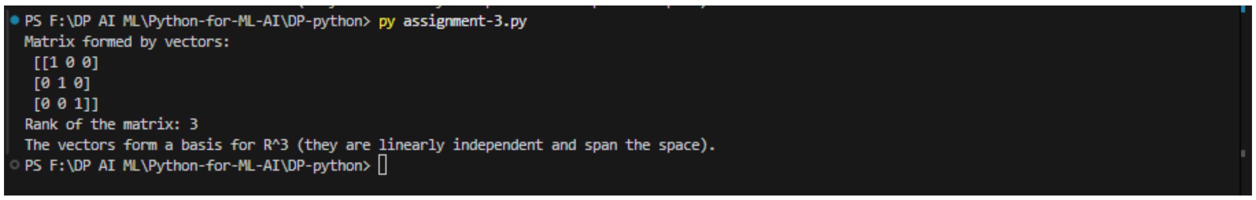
Basis of a vector space	Basis of a column space
The smallest set of vectors that can build the entire space.	The smallest set of matrix columns that span the column subspace
No, always equals the space’s dimension	Yes, especially when columns are linearly dependent
Used to describe the structure of a whole vector space	Used to analyze solutions to linear systems from a matrix
Entire spaces like R^2, R^3…etc.	A specific matrix and its columns .

12. Use NumPy to check whether the following vectors form a basis for R: v1 = [1, 0, 0] v2 = [0, 1, 0] v3 = [0, 0, 1].

```
import numpy as np

v1 = [1, 0, 0]
v2 = [0, 1, 0]
v3 = [0, 0, 1]
```

```python
A = np.column_stack((v1, v2, v3))


rank = np.linalg.matrix_rank(A)

print("Matrix formed by vectors:\n", A)
print("Rank of the matrix:", rank)

if rank == 3:
    print("The vectors form a basis for R^3 (they are linearly independent and
span the space).")
else:
    print("The vectors do NOT form a basis for R^3.")
```

Output:

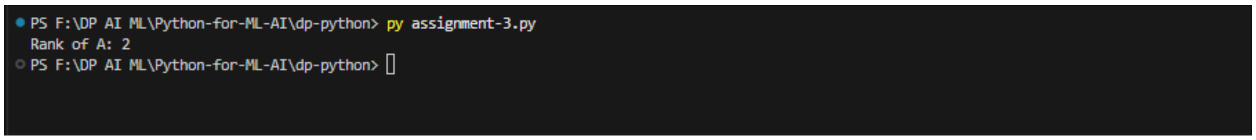


```
PS F:\DP AI ML\Python-for-ML-AI\DP-python> py assignment-3.py
Matrix formed by vectors:
 [[1 0 0]
 [0 1 0]
 [0 0 1]]
Rank of the matrix: 3
The vectors form a basis for R^3 (they are linearly independent and span the space).
PS F:\DP AI ML\Python-for-ML-AI\DP-python>
```

14. Compute the rank of the following matrix using NumPy:

```python
import numpy as np

A = np.array([
    [1, 2, 3],
    [2, 4, 6],
    [1, 1, 1]
])

rank = np.linalg.matrix_rank(A)
print("Rank of A:", rank)
```

Output:



```
PS F:\DP AI ML\Python-for-ML-AI\dp-python> py assignment-3.py
Rank of A: 2
PS F:\DP AI ML\Python-for-ML-AI\dp-python>
```

Row 1 and row 3 are linearly independent and row 2 is dependent, so matrix has only two linearly independent so hence rank is less then 3.

15. Prove that the rank of a matrix equals the number of pivot columns in its row echelon form. Illustrate with an example matrix.

A pivot column is defined as that column which contains the leading entry in a particular row of matrix's row echelon form.

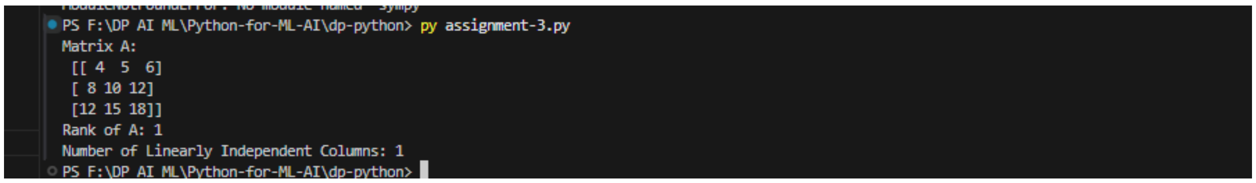
- ✓ When a matrix is reduced to its row echelon form by means of elementary row operations, thereby obtaining the following in every non-zero row, there will be a leading 1 (pivot) in some column.
- ✓ The total number of such positions equals the total number of linearly independent rows or columns.
- ✓ Hence, the number of pivot columns is equal to the rank of the matrix.

```python
import numpy as np
from sympy import Matrix

A = np.array([
    [1, 2, 3],
    [2, 4, 6],
    [1, 1, 1]
])

rank_numpy = np.linalg.matrix_rank(A)

A_sym = Matrix(A)
ref, _ = A_sym.rref()

pivot_columns = _.tolist()
pivot_count = len(pivot_columns)

print("Row Echelon Form (RREF):")
print(ref)
print("Pivot Columns:", pivot_columns)
print("Number of Pivot Columns:", pivot_count)
print("Rank from NumPy:", rank_numpy)
```

16. Construct a 3×3 matrix of rank 1. Use NumPy to verify that it has only one linearly independent column

```python
import numpy as np


u = np.array([1, 2, 3])
v = np.array([4, 5, 6])


A = np.outer(u, v)

rank = np.linalg.matrix_rank(A)

_, s, _ = np.linalg.svd(A)
independent_columns = np.sum(s > 1e-10)

print("Matrix A:\n", A)
print("Rank of A:", rank)
print("Number of Linearly Independent Columns:", independent_columns)
```

Output:

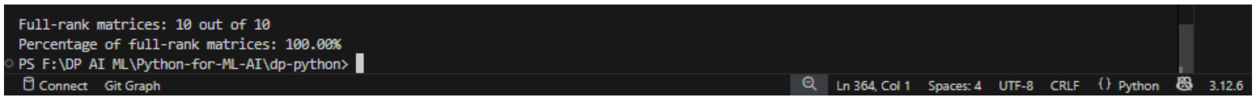


```
PS F:\DP AI ML\Python-for-ML-AI\dp-python> py assignment-3.py
Matrix A:
 [[ 4  5  6]
 [ 8 10 12]
 [12 15 18]]
Rank of A: 1
Number of Linearly Independent Columns: 1
PS F:\DP AI ML\Python-for-ML-AI\dp-python>
```

17. Write a Python function to generate 10 random 4×4 matrices. For each, compute its rank and determine how many are full rank. Report the percentage.

```python
import numpy as np

def full_rank_percentage(n=10, size=4):
    full_rank_count = 0

    for i in range(n):
        A = np.random.randint(-10, 10, (size, size))
        rank = np.linalg.matrix_rank(A)
        print(f"Matrix {i+1}:\n{A}\nRank: {rank}\n")
        if rank == size:
            full_rank_count += 1

    percentage = (full_rank_count / n) * 100
    print(f"Full-rank matrices: {full_rank_count} out of {n}")
    print(f"Percentage of full-rank matrices: {percentage:.2f}%")
    return percentage


full_rank_percentage()
```

Output:

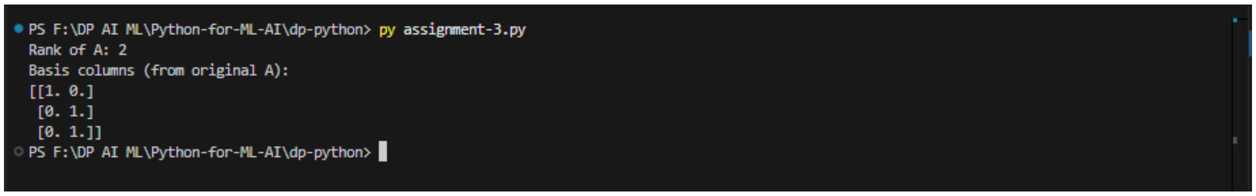


```
Full-rank matrices: 10 out of 10
Percentage of full-rank matrices: 100.00%
PS F:\DP AI ML\Python-for-ML-AI\dp-python>
```

19. Find the rank of A and determine which columns form a basis for its column space.

```python
import numpy as np

A = np.array([
    [1, 0, 1],
    [0, 1, 1],
    [0, 1, 1]
], dtype=float)

rank = np.linalg.matrix_rank(A)
print("Rank of A:", rank)


u, s, vh = np.linalg.svd(A)
tolerance = 1e-10
independent_indices = np.where(s > tolerance)[0]

basis_columns = A[:, independent_indices]

print("Basis columns (from original A):")
print(basis_columns)
```

Output:

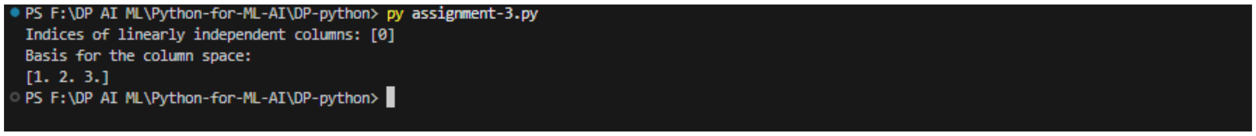


```
PS F:\DP AI ML\Python-for-ML-AI\dp-python> py assignment-3.py
Rank of A: 2
Basis columns (from original A):
[[1. 0.]
 [0. 1.]
 [0. 1.]]
PS F:\DP AI ML\Python-for-ML-AI\dp-python>
```

20. Write a Python script that takes any matrix as input and outputs a set of linearly independent columns (i.e., a basis for the column space).

```python
import numpy as np

def get_column_basis_numpy(A, tol=1e-10):

    A = np.array(A, dtype=float)
    Q, R = np.linalg.qr(A)
    diag = np.abs(np.diag(R))


    independent_indices = [i for i, val in enumerate(diag) if val > tol]
    basis_columns = [A[:, i] for i in independent_indices]

    return basis_columns, independent_indices


def main():
    A = np.array([
        [1, 2, 3],
        [2, 4, 6],
        [3, 6, 9]
    ])

    basis, indices = get_column_basis_numpy(A)

    print("Indices of linearly independent columns:", indices)
    print("Basis for the column space:")
    for col in basis:
        print(col)


main()
```

Output:



```
PS F:\DP AI ML\Python-for-ML-AI\DP-python> py assignment-3.py
Indices of linearly independent columns: [0]
Basis for the column space:
[1. 2. 3.]
PS F:\DP AI ML\Python-for-ML-AI\DP-python>
```