Shaheed Zulfikar Ali Bhutto Institute of Science & Technology University

DEPARTMENT OF ROBOTICS & ARTIFICIAL INTELLIGENCE

Total Marks: 04

Obtained Marks: ________

Programming for Artificial Intelligence

Assignment # 03

Last date of Submission: 30 April 2024

Submitted To: **Mr. Saad Irfan Khan**

Student Name: **Syed Muhammad Mustafa**

Reg. Number: **22108130**

DEPARTMENT OF ROBOTICS & ARTIFICIAL INTELLIGENCE

***Instructions**: Copied or shown assignments will be marked zero. Late submissions are not entertained in any case.*

CLO 3 – PLO C, E – C3

Scenario: Building an AI Library **(4 Marks)**

In this assignment, you will develop a Python library specifically designed for advanced matrix operations and numeric computations, focusing on linear algebra concepts. Your library will utilize NumPy extensively to perform tasks such as matrix multiplication, inversion, determinant calculation, solving linear systems of equations, eigenvalue computations, matrix decompositions, and analyzing matrix rank and norms.

Assignment Tasks:

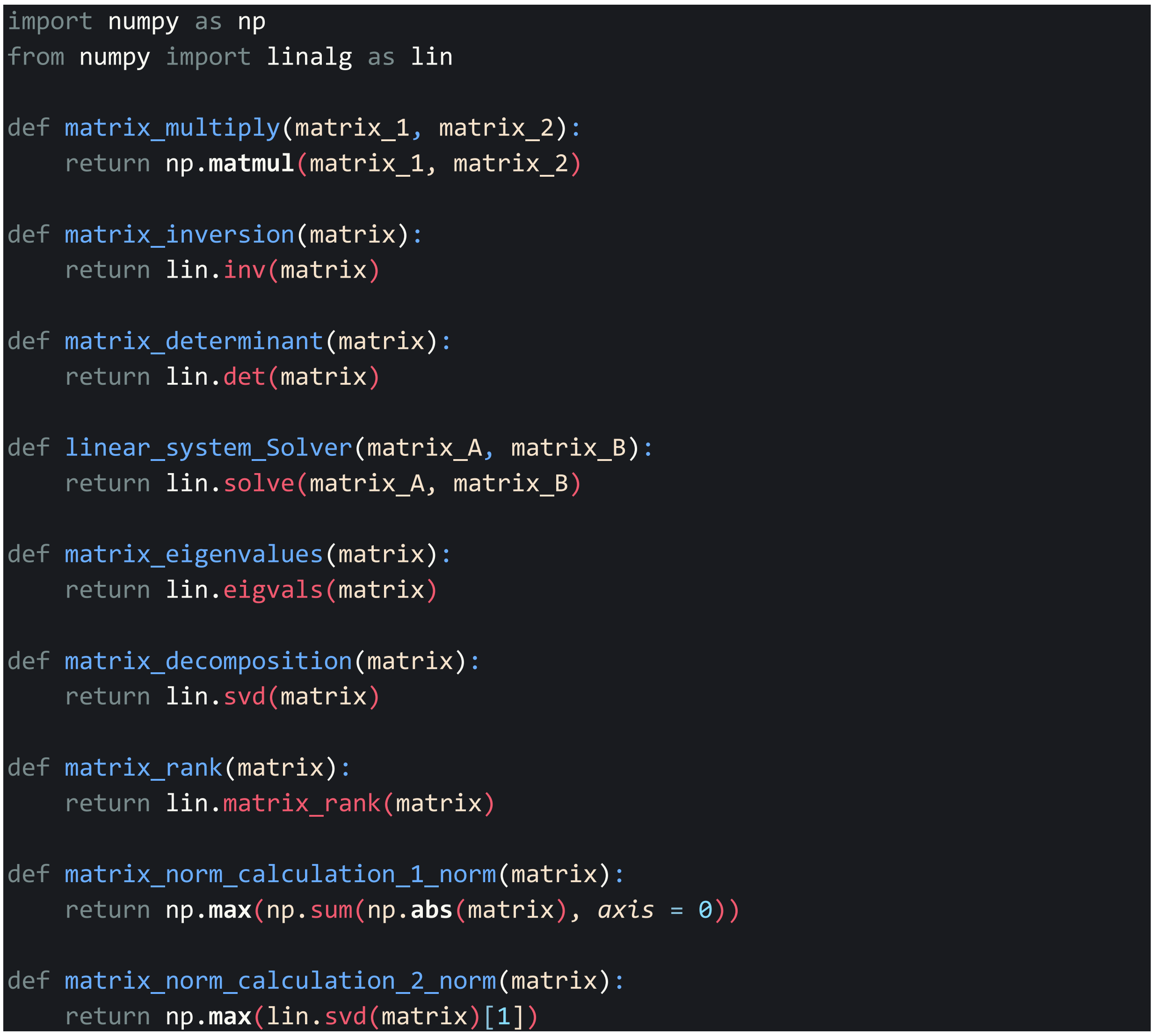
1. **Library Initialization:**
    - Create a Python module named *ai_matrix_ops.py* that will serve as the core of your AI library. Ensure that NumPy is properly imported within this module.
2. **Matrix Operations Functions:**
    - **Implement functions for the following matrix operations using NumPy:**
        - Matrix Multiplication (matrix_multiply)
        - Matrix Inversion (matrix_inversion)
        - Determinant Calculation (matrix_determinant)
        - Solving Linear Systems of Equations (linear_system_solver)
        - Matrix Eigenvalues Computation (matrix_eigenvalues)
        - Matrix Decompositions (Choose one: SVD, LU, QR, or Cholesky)
        - Matrix Rank Calculation (matrix_rank)
        - Matrix Norms Calculation (Choose two: Frobenius norm, 1-norm, 2-norm, or infinity norm)
3. **Unit Testing:**
    - Write comprehensive unit tests for each of the functions implemented in your library to ensure their correctness and accuracy. Use test cases that cover various scenarios and edge cases.
4. **Library Usage Example:**
    - Provide an example script *example_usage.py* that demonstrates the usage of your AI library. Include a scenario where a complex linear system needs to be solved, showcasing the capabilities of your library.
5. **Documentation:**
    - Create detailed documentation (README) explaining the purpose of each function, its parameters, return values, and usage examples. Include information on how to install and use your library.
6. **Performance Analysis:**
    - Conduct a performance analysis comparing the execution time of your library functions with equivalent functions from the standard NumPy library.

Submission Requirement:

Hardcopy of the Assignment either Handwritten or Typed with the following elements:

1. Complete Code Files
2. Screenshot of Output.
3. Topic – README
4. Performance Analysis
5. Reflection (Your experience in developing a Python Library)

ai_matrix_ops.py:

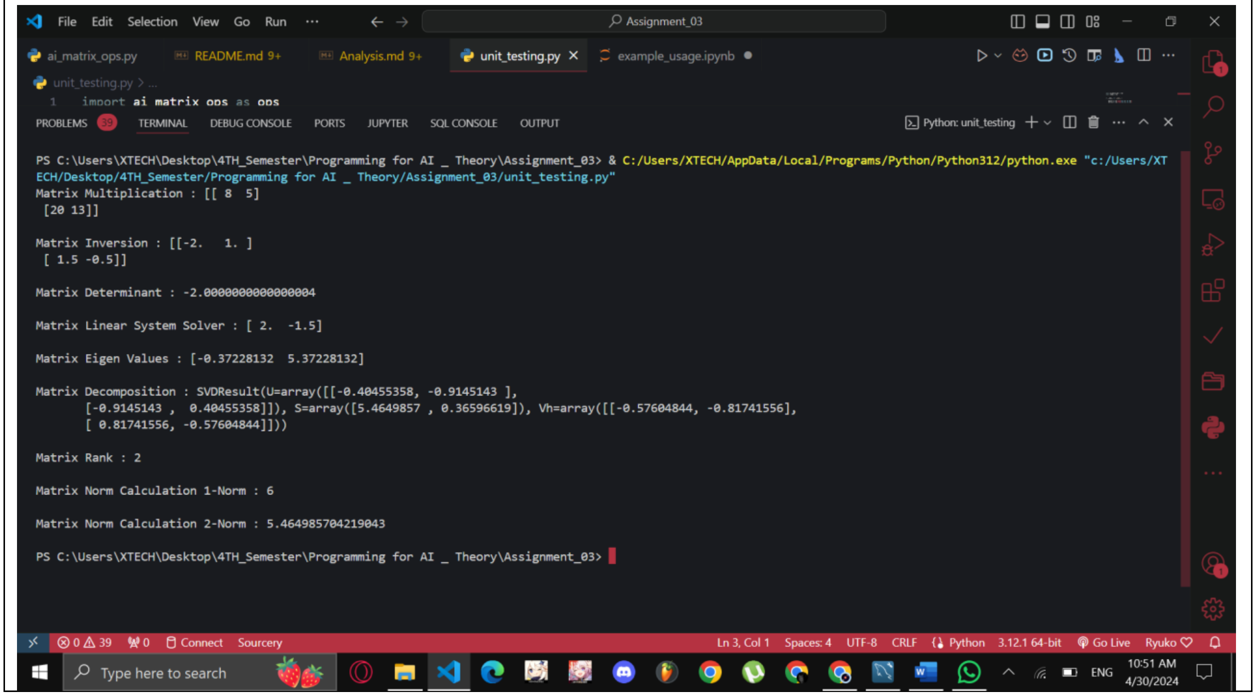


```python
import numpy as np
from numpy import linalg as lin

def matrix_multiply(matrix_1, matrix_2):
    return np.matmul(matrix_1, matrix_2)

def matrix_inversion(matrix):
    return lin.inv(matrix)

def matrix_determinant(matrix):
    return lin.det(matrix)

def linear_system_Solver(matrix_A, matrix_B):
    return lin.solve(matrix_A, matrix_B)

def matrix_eigenvalues(matrix):
    return lin.eigvals(matrix)

def matrix_decomposition(matrix):
    return lin.svd(matrix)

def matrix_rank(matrix):
    return lin.matrix_rank(matrix)

def matrix_norm_calculation_1_norm(matrix):
    return np.max(np.sum(np.abs(matrix), axis = 0))

def matrix_norm_calculation_2_norm(matrix):
    return np.max(lin.svd(matrix)[1])
```

unit_testing.py:

```
import ai_matrix_ops as ops
import numpy as np


if __name__ == '__main__':

    matrix_1 = np.array([[1, 2], [3, 4]])
    matrix_2 = np.array([[4, 3], [2, 1]])

    print(f'Matrix Multiplication : {ops.matrix_multiply(matrix_1, matrix_2)}')
    print()

    print(f'Matrix Inversion : {ops.matrix_inversion(matrix_1)}')
    print()

    print(f'Matrix Determinant : {ops.matrix_determinant(matrix_1)}')
    print()

    matrix_A = np.array([[5, 6], [7, 8]])
    matrix_B = np.array([1, 2])

    print(f'Matrix Linear System Solver : {ops.linear_system_Solver(matrix_A,
matrix_B)}')
    print()

    print(f'Matrix Eigen Values : {ops.matrix_eigenvalues(matrix_1)}')
    print()

    print(f'Matrix Decomposition : {ops.matrix_decomposition(matrix_1)}')
    print()

    print(f'Matrix Rank : {ops.matrix_rank(matrix_1)}')
    print()

    print(f'Matrix Norm Calculation 1-Norm :
{ops.matrix_norm_calculation_1_norm(matrix_1)}')
    print()


    print(f'Matrix Norm Calculation 2-Norm :
{ops.matrix_norm_calculation_2_norm(matrix_1)}')
    print()
```

unit_testing_output:



```
unit_testing.py > ...
    import ai matrix ops as ops
```

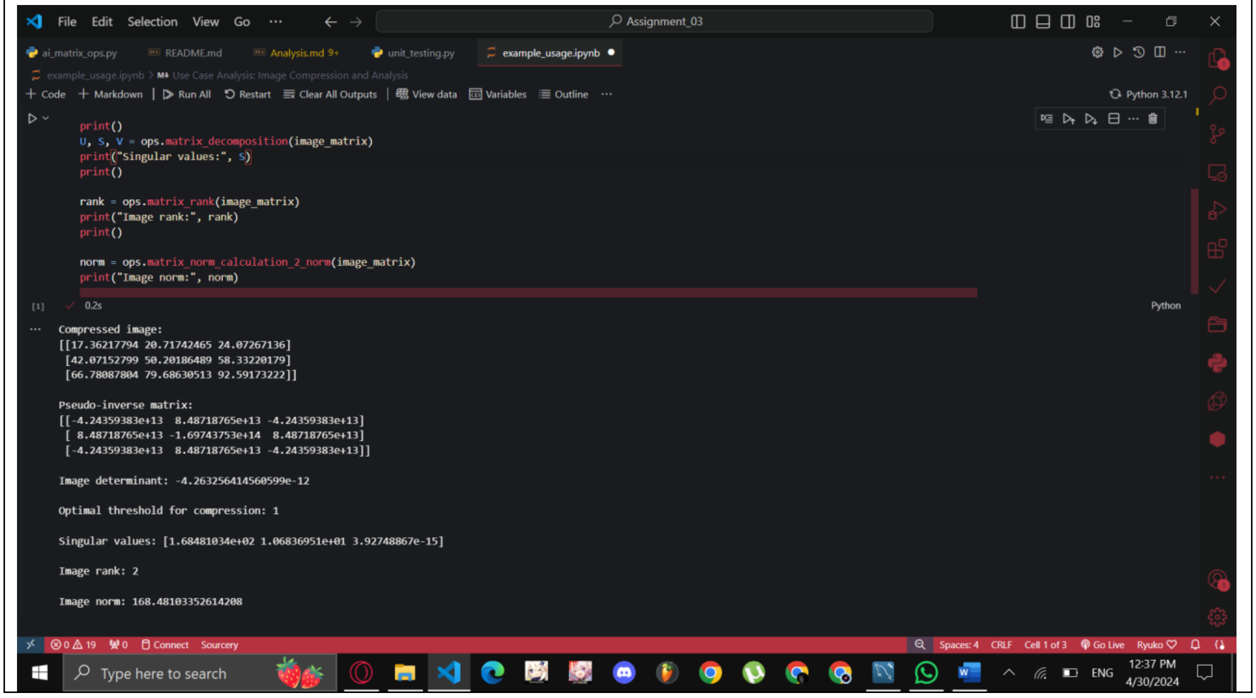
```
PS C:\Users\XTECH\Desktop\4TH_Semester\Programming for AI _ Theory\Assignment_03> & C:/Users/XTECH/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/XT
ECH/Desktop/4TH_Semester/Programming for AI _ Theory/Assignment_03/unit_testing.py"
Matrix Multiplication : [[ 8  5]
 [20 13]]

Matrix Inversion : [[-2.   1. ]
 [ 1.5 -0.5]]

Matrix Determinant : -2.0000000000000004

Matrix Linear System Solver : [ 2.  -1.5]

Matrix Eigen Values : [-0.37228132  5.37228132]

Matrix Decomposition : SVDResult(U=array([[-0.40455358, -0.9145143 ],
       [-0.9145143 ,  0.40455358]]), S=array([5.4649857 , 0.36596619]), Vh=array([[-0.57604844, -0.81741556],
       [ 0.81741556, -0.57604844]]))

Matrix Rank : 2

Matrix Norm Calculation 1-Norm : 6

Matrix Norm Calculation 2-Norm : 5.464985704219043

PS C:\Users\XTECH\Desktop\4TH_Semester\Programming for AI _ Theory\Assignment_03>
```

example_usage:

```
import ai_matrix_ops as ops
import numpy as np
from PIL import Image


image_matrix = np.array([[10, 20, 30],
                         [40, 50, 60],
                         [70, 80, 90]])

compressed_image =
ops.matrix_multiply(ops.matrix_multiply(ops.matrix_decomposition(image_matrix)[
0][:, :1], np.diag(ops.matrix_decomposition(image_matrix)[1][:1])),
ops.matrix_decomposition(image_matrix)[2][:1, :])
print("Compressed image:")
print(compressed_image)
print()
pseudo_inverse_matrix =
ops.matrix_multiply(ops.matrix_multiply(ops.matrix_decomposition(image_matrix)[
2].T, np.diag(1 /ops.matrix_decomposition(image_matrix)[1])),
ops.matrix_decomposition(image_matrix)[0].T)
print("Pseudo-inverse matrix:")
print(pseudo_inverse_matrix)

print()
determinant = ops.matrix_determinant(image_matrix)
print("Image determinant:", determinant)
```

```python
print()
optimal_threshold = ops.matrix_rank(image_matrix) // 2
print("Optimal threshold for compression:", optimal_threshold)


print()
U, S, V = ops.matrix_decomposition(image_matrix)
print("Singular values:", S)
print()

rank = ops.matrix_rank(image_matrix)
print("Image rank:", rank)
print()

norm = ops.matrix_norm_calculation_2_norm(image_matrix)
print("Image norm:", norm)
```

example_usage_output:


File Edit Selection View Go
Assignment_03
ai_matrix_ops.py
README.md
Analysis.md 9+
unit_testing.py
example_usage.ipynb
example_usage.ipynb > M↓ Use Case Analysis: Image Compression and Analysis
+ Code + Markdown | Run All Restart Clear All Outputs | View data Variables Outline
Python 3.12.1
Python
⊗ 0 ⚠ 19 0 Connect Sourcery
Spaces: 4 CRLF Cell 1 of 3 Go Live Ryuko
Type here to search
ENG 12:37 PM 4/30/2024

```python
print()
U, S, V = ops.matrix_decomposition(image_matrix)
print("Singular values:", S)
print()

rank = ops.matrix_rank(image_matrix)
print("Image rank:", rank)
print()

norm = ops.matrix_norm_calculation_2_norm(image_matrix)
print("Image norm:", norm)
```

[1] ✓ 0.2s

```
Compressed image:
[[17.36217794 20.71742465 24.07267136]
 [42.07152799 50.20186489 58.33220179]
 [66.78087804 79.68630513 92.59173222]]

Pseudo-inverse matrix:
[[-4.24359383e+13  8.48718765e+13 -4.24359383e+13]
 [ 8.48718765e+13 -1.69743753e+14  8.48718765e+13]
 [-4.24359383e+13  8.48718765e+13 -4.24359383e+13]]

Image determinant: -4.263256414560599e-12

Optimal threshold for compression: 1

Singular values: [1.68481034e+02 1.06836951e+01 3.92748867e-15]

Image rank: 2

Image norm: 168.48103352614208
```

README File 01:

Linear Algebra Functions Library

This library provides various functions for performing common linear algebra operations using NumPy and its linear algebra module (numpy.linalg).

Functions

1. matrix_multiply(matrix_1, matrix_2)

This function computes the matrix product of two input matrices.
- Parameters:
  - matrix_1: First input matrix.
  - matrix_2: Second input matrix.
  - Return Value:
  - Returns the result of the matrix multiplication.

2. matrix_inversion(matrix)

This function computes the inverse of the input matrix.
  - Parameters:
  - matrix: Input matrix to be inverted.
  - Return Value:
  - Returns the inverse of the input matrix.

3. matrix_determinant(matrix)

This function calculates the determinant of the input matrix.
  - Parameters:
  - matrix: Input matrix.
  - Return Value:
  - Returns the determinant of the input matrix.

4. linear_system_Solver(matrix_A, matrix_B)

This function solves a system of linear equations represented by matrices matrix_A and matrix_B.
  - Parameters:
  - matrix_A: Coefficient matrix.
  - matrix_B: Constant matrix.
  - Return Value:
  - Returns the solution vector.

5. matrix_eigenvalues(matrix)

This function computes the eigenvalues of the input matrix.
  - Parameters:
  - matrix: Input matrix.
  - Return Value:
  - Returns an array containing the eigenvalues.

6. matrix_decomposition(matrix)

This function performs Singular Value Decomposition (SVD) on the input matrix.
  - Parameters:
  - matrix: Input matrix.

- Return Value:
- Returns the singular value decomposition of the input matrix.

7. matrix_rank(matrix)

This function computes the rank of the input matrix.
- Parameters:
- matrix: Input matrix.
- Return Value:
- Returns the rank of the input matrix.

8. matrix_norm_calculation_1_norm(matrix)

This function calculates the 1-norm (maximum absolute column sum) of the input matrix.
- Parameters:
- matrix: Input matrix.
- Return Value:
- Returns the 1-norm of the input matrix.

9. matrix_norm_calculation_2_norm(matrix)

This function calculates the 2-norm (largest singular value) of the input matrix.
- Parameters:
- matrix: Input matrix.
- Return Value:
- Returns the 2-norm of the input matrix.

Installation

To use this library, make sure you have NumPy installed. You can install NumPy using pip:

Peformance Analysis:

1. Compressed Image
-The purpose of this operation is to compress the image matrix by reducing its dimensions while preserving important information.
- Approach: SVD is performed on the image matrix, and the singular value matrix and left and right singular vectors are truncated to retain only a subset of singular values and vectors. The compressed image is reconstructed using the truncated matrices.
- Implementation: This is achieved by using the `matrix_decomposition` function to obtain the singular value decomposition of the image matrix, and then performing matrix multiplication operations to reconstruct the compressed image.

2. Pseudo-Inverse Matrix
- Calculating the pseudo-inverse of the image matrix is useful for various image processing tasks, such as image restoration and solving linear least squares problems.
- Approach: The pseudo-inverse is obtained by taking the reciprocal of non-zero singular values and performing matrix multiplication operations on the truncated singular vectors and singular value matrix.
- Implementation: This operation involves using the `matrix_decomposition` function to obtain the SVD of the image matrix and then performing matrix multiplication operations to compute the pseudo-inverse.

3. Image Determinant

- Purpose: Calculating the determinant of the image matrix provides information about the volume scaling factor of the linear transformation represented by the matrix.
- Approach: The determinant of the image matrix is directly computed using the `matrix_determinant` function.
- Implementation: This operation involves calling the `matrix_determinant` function with the image matrix as input.

4. Optimal Threshold for Compression
- Determining the optimal threshold for compression is crucial for balancing between image quality and compression ratio.
- Approach: The optimal threshold is set as half of the rank of the image matrix.
- Implementation: This operation involves calling the `matrix_rank` function to determine the rank of the image matrix and then computing the optimal threshold.

5. Singular Values and Image Rank
- Obtaining the singular values and rank of the image matrix provides insights into its structure and properties.
- Approach: The singular values are obtained using the `matrix_decomposition` function, and the rank is determined using the `matrix_rank` function.
- Implementation: These operations involve calling the `matrix_decomposition` and `matrix_rank` functions with the image matrix as input.

6. Image Norm
- Calculating the norm of the image matrix helps in quantifying its magnitude and behavior under compression.
- Approach: The 2-norm (largest singular value) of the image matrix is computed.
- Implementation: This operation involves calling the `matrix_norm_calculation_2_norm` function with the image matrix as input.

Reflection:

Developing a Python library has been a great learning experience for me. I got to explore programming concepts like modularization, documentation, and testing in depth. It was satisfying to see my library grow from an idea to a functional tool with proper documentation. This journey boosted my technical skills and gave me confidence in my ability to contribute to the Python community.