Practical of mathematics for computing

RAMANUJAN COLLEGE

UNIVERSITY OF DELHI

DSC 03: *MATHEMATICS FOR COMPUTING-1*

SEMESTER: 1

(2024-2025)

Submitted by-

Name - Radhika
College roll no. - 24570048
University roll.no. - 24020570050
Course -
B.Sc. (Hons.) Computer Science
Semester 1

Submitted to-

Dr. Aakash
Assistant Professor (Operational Research),
Department of Computer Science,
Ramanujan College,
University of Delhi,
CR Park Main Road,
Block H, Kalkaji,
New Delhi-110019

INDEX

S.NO.	PRACTICALS	Page no.	signature
01	Find cofactor, determinant, adjoint and inverse of a matrix.	4 -5	
02	Convert a matrix into echelon form and find its rank.	6 -7	
03	Solve a system of equations using the gauss elimination method.	8 -9	
04	Solve a system of equations wing the Gauss Jordan method.	10-11	
05	Verify the linear dependence of vectors generate a linear combination of given vectors of Rn/matrices of the same size.	12-13	
06	Check the diagonalizable property of matrices and find the corresponding eigenvalue and verify the Cayley Hamilton theorem.	14-15	
07	Compute Gradient of a scalar field, Divergence and Curl of a vector filed.	16-17	

ACKNOWLEDGEMENT

I would like to express my heartfelt gratitude to Dr. Aakash, Assistant Professor (Operational Research), Ramanujan College, University of Delhi, for providing me with the opportunity to work on this practical file for Mathematics for Computing 1. His guidance and encouragement have been invaluable throughout the preparation of this file.

I am also thankful to my peers and the faculty members of the Computer Science Department for their continuous support and valuable insights, which have greatly. enriched my understanding of the subject.

Finally, I extend my sincere appreciation to my family and friends for their constant motivation and encouragement, which

has been a source of strength in completing this work successfully.

RADHIKA

24020570050

PRACTICAL -01

#find cofactors, determinant, adjoint, and inverse of a matrix.

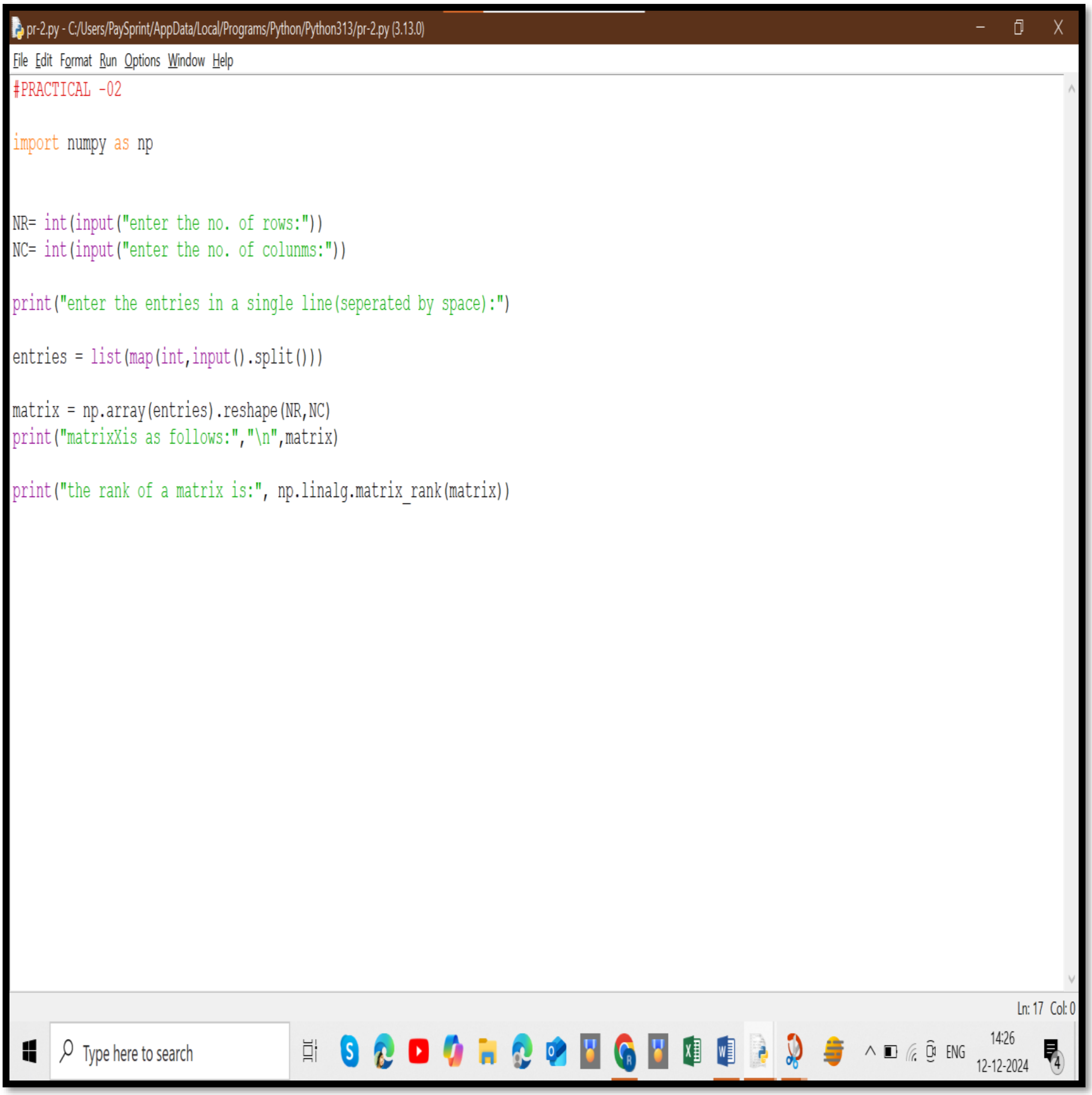
```python
import numpy as np

NR=int(input("Enter the no. of rows:"))

NC=int(input("Enter the no. of columns: "))

print("Enter the entries in a single line (separated by space):")

entries =list(map(int,input().split()))

A= np.array(entries).reshape(NR, NC)
print("matrix A is as follows:","\n",A)

A_Inverse=np.linalg.inv(A)

print("inverse of  A is" ,"\n",A_Inverse)

Transpose_of_A_Inverse=np.transpose(A_Inverse)

print("Transpose of A Inverse is","\n", Transpose_of_A_Inverse)

Determinant_of_A=np.linalg.det(A)

print("Determinant of A","\n", Determinant_of_A)


Cofactor_of_A=np.dot(Transpose_of_A_Inverse, Determinant_of_A)

print("The Cofactor of a Matrix is:", "\n",Cofactor_of_A)

Adjoint_of_A=np.transpose( Cofactor_of_A)

print("The Ajoint of a Matrix is:","\n",Adjoint_of_A)
```

OUTPUT

```
Python 3.13.0 (tags/v3.13.0:60403a5, Oct  7 2024, 09:38:07) [MSC v.1941 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/PaySprint/AppData/Local/Programs/Python/Python313/pr-01.py =
Enter the no. of rows:3
Enter the no. of columns: 3
Enter the entries in a single line (separated by space):
2 3 5 6 8 9 5 7 3
matrix A is as follows:
 [[2 3 5]
 [6 8 9]
 [5 7 3]]
inverse of  A is
 [[-3.          2.         -1.        ]
 [ 2.07692308 -1.46153846  0.92307692]
 [ 0.15384615  0.07692308 -0.15384615]]
Transpose of A Inverse is
 [[-3.          2.07692308  0.15384615]
 [ 2.         -1.46153846  0.07692308]
 [-1.          0.92307692 -0.15384615]]
Determinant of A
 13.000000000000005
The Cofactor of a Matrix is:
 [[-39.  27.   2.]
 [ 26. -19.   1.]
 [-13.  12.  -2.]]
The Ajoint of a Matrix is:
 [[-39.  26. -13.]
 [ 27. -19.  12.]
 [  2.   1.  -2.]]
>>>
```

PRACTICAL -02

#convert the matrix into echelon form and find its rank.



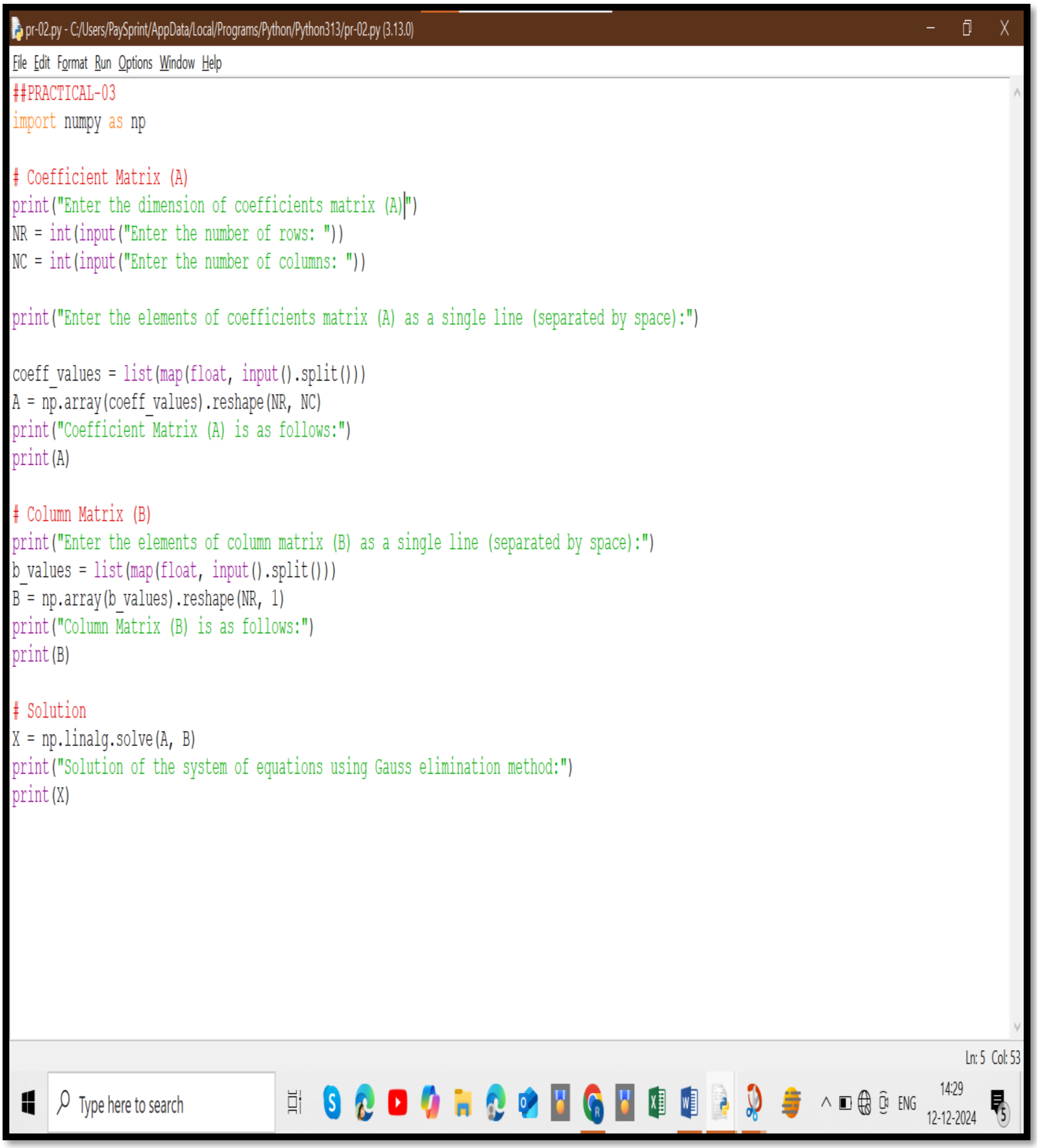

```
#PRACTICAL -02

import numpy as np


NR= int(input("enter the no. of rows:"))
NC= int(input("enter the no. of colunms:"))

print("enter the entries in a single line(seperated by space):")

entries = list(map(int,input().split()))

matrix = np.array(entries).reshape(NR,NC)
print("matrixXis as follows:","\n",matrix)

print("the rank of a matrix is:", np.linalg.matrix_rank(matrix))
```

OUTPUT

```
IDLE Shell 3.13.0
File Edit Shell Debug Options Window Help
Python 3.13.0 (tags/v3.13.0:60403a5, Oct  7 2024, 09:38:07) [MSC v.1941 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
== RESTART: C:\Users\PaySprint\AppData\Local\Programs\Python\Python313\pr-2.py =
enter the no. of rows:3
enter the no. of colunms:3
enter the entries in a single line(seperated by space):
5 7 9 0 6 4 8 9 3
matrix X is as follows:
 [[5 7 9]
 [0 6 4]
 [8 9 3]]
the rank of a matrix is: 3
>>>
Ln: 14 Col: 0
```

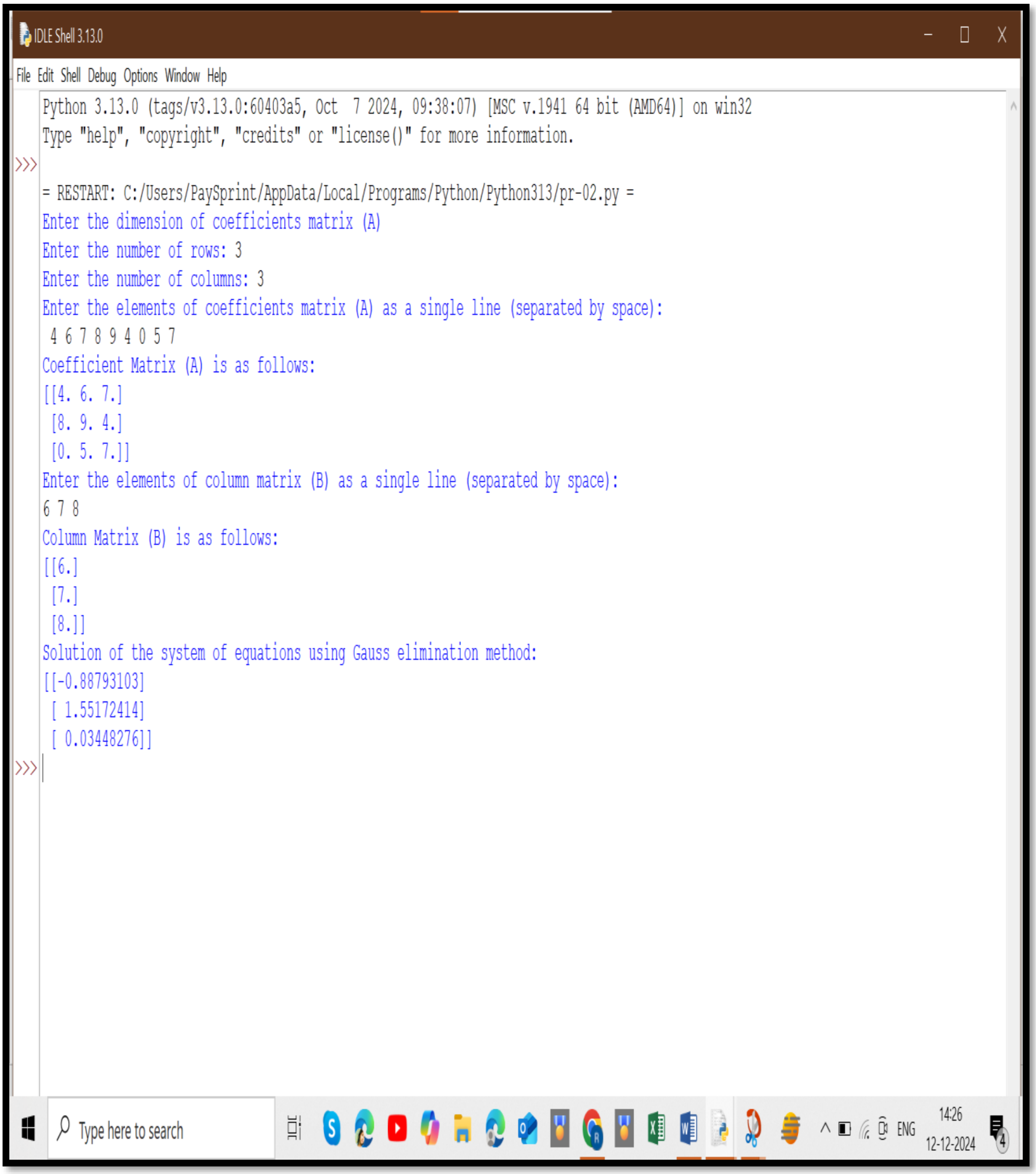
PRACTICAL -03

solve a system of equations using gauss elimination method.




```python
##PRACTICAL-03
import numpy as np

# Coefficient Matrix (A)
print("Enter the dimension of coefficients matrix (A)")
NR = int(input("Enter the number of rows: "))
NC = int(input("Enter the number of columns: "))

print("Enter the elements of coefficients matrix (A) as a single line (separated by space):")

coeff_values = list(map(float, input().split()))
A = np.array(coeff_values).reshape(NR, NC)
print("Coefficient Matrix (A) is as follows:")
print(A)

# Column Matrix (B)
print("Enter the elements of column matrix (B) as a single line (separated by space):")
b_values = list(map(float, input().split()))
B = np.array(b_values).reshape(NR, 1)
print("Column Matrix (B) is as follows:")
print(B)

# Solution
X = np.linalg.solve(A, B)
print("Solution of the system of equations using Gauss elimination method:")
print(X)
```

OUTPUT


IDLE Shell 3.13.0
File Edit Shell Debug Options Window Help
Python 3.13.0 (tags/v3.13.0:60403a5, Oct 7 2024, 09:38:07) [MSC v.1941 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/PaySprint/AppData/Local/Programs/Python/Python313/pr-02.py =
Enter the dimension of coefficients matrix (A)
Enter the number of rows: 3
Enter the number of columns: 3
Enter the elements of coefficients matrix (A) as a single line (separated by space):
4 6 7 8 9 4 0 5 7
Coefficient Matrix (A) is as follows:
[[4. 6. 7.]
[8. 9. 4.]
[0. 5. 7.]]
Enter the elements of column matrix (B) as a single line (separated by space):
6 7 8
Column Matrix (B) is as follows:
[[6.]
[7.]
[8.]]
Solution of the system of equations using Gauss elimination method:
[[-0.88793103]
[1.55172414]
[0.03448276]]
>>>
Type here to search
ENG
14:26
12-12-2024

PRACTICAL -04

solve a system of equations using gauss Jordan method.

```python
#PRACTICAL-04
import numpy as np

# Coefficient Matrix (A)
print("Enter the dimension of coefficients matrix (A)")
NR = int(input("Enter the number of rows: "))
NC = int(input("Enter the number of columns: "))

print("Enter the elements of coefficients matrix (A) as a single line (separated by space):")

coefficient_entries = list(map(float, input().split()))
coefficient_matrix = np.array(coefficient_entries).reshape(NR, NC)
print("Coefficient Matrix (A) is as follows:\n",coefficient_matrix,"\n")

# Column Matrix (B)
print("Enter the elements of column matrix (B) as a single line (separated by space):")
column_entries = list(map(float, input().split()))
column_matrix = np.array(column_entries).reshape(NR, 1)
print("Column Matrix (B) is as follows:\n",column_matrix,"\n")

# Solution
inv_ofcoefficient_matrix = np.linalg.inv(coefficient_matrix)
solution_of_the_system_of_equations = np.matmul(coefficient_matrix,column_matrix)
print("Solution of the system of equations using Gauss jordan method:")
print(solution_of_the_system_of_equations)
```

OUTPUT

```
IDLE Shell 3.13.0
File  Edit  Shell  Debug  Options  Window  Help
Python 3.13.0 (tags/v3.13.0:60403a5, Oct  7 2024, 09:38:07) [MSC v.1941 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/PaySprint/AppData/Local/Programs/Python/Python313/pr-04.py =
Enter the dimension of coefficients matrix (A)
Enter the number of rows: 3
Enter the number of columns: 3
Enter the elements of coefficients matrix (A) as a single line (separated by space):
3 6 8 2 0 1 9 4 7
Coefficient Matrix (A) is as follows:
 [[3. 6. 8.]
 [2. 0. 1.]
 [9. 4. 7.]]

Enter the elements of column matrix (B) as a single line (separated by space):
3 8 6
Column Matrix (B) is as follows:
 [[3.]
 [8.]
 [6.]]

Solution of the system of equations using Gauss jordan method:
[[105.]
 [ 12.]
 [101.]]
>>>
Ln: 8  Col: 84
```

PRACTICAL -05

#Verify the linear dependence of vectors generate a linear combination of given vectors of Rn/matrices of the same size.

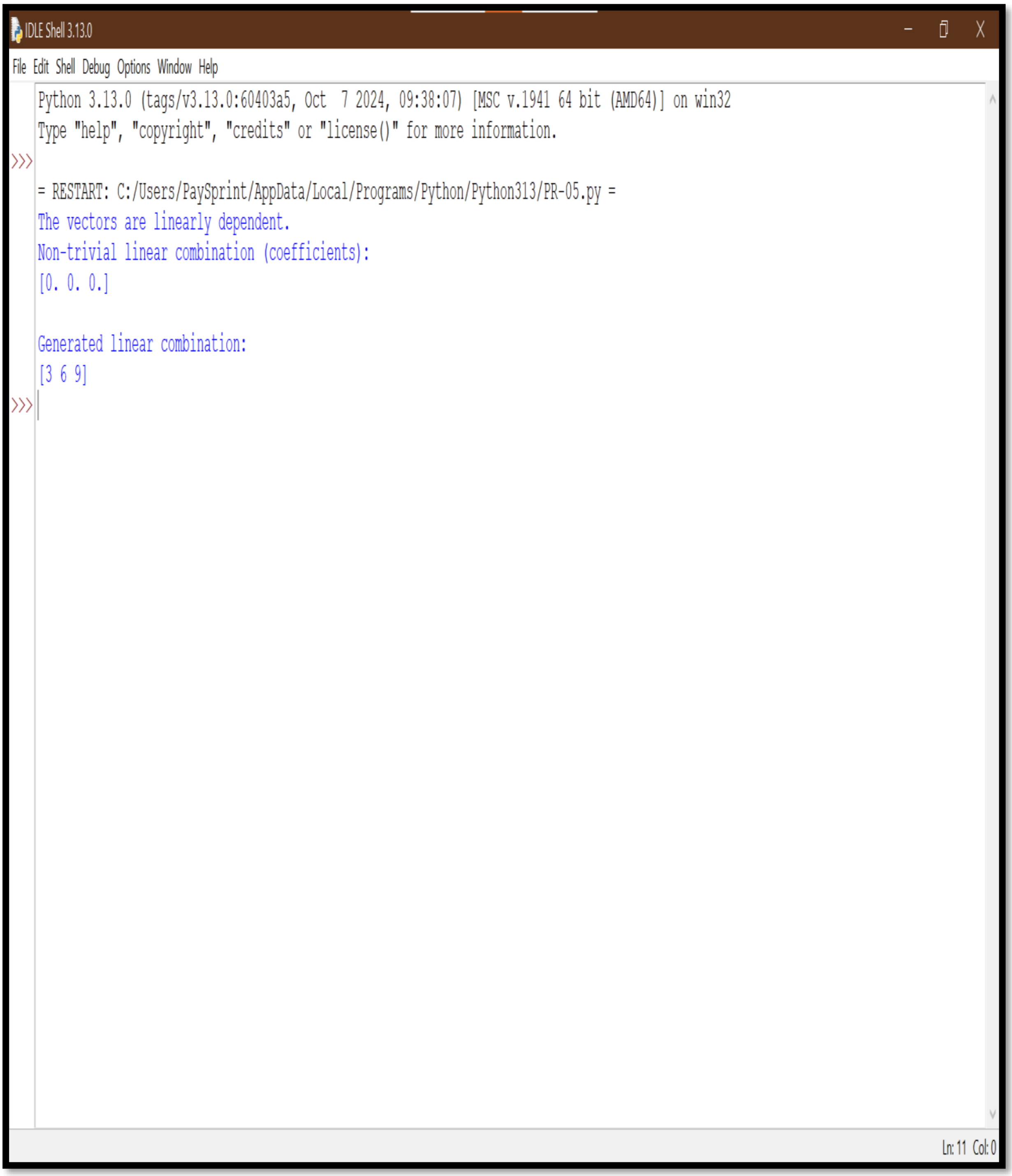
PR-05.py - C:/Users/PaySprint/AppData/Local/Programs/Python/Python313/PR-05.py (3.13.0)

File Edit Format Run Options Window Help

```python
#PRACTICAL-05
import numpy as np

def is_linearly_dependent(vectors):
    # Convert the list of vectors to a numpy matrix (each vector is a column)
    A = np.column_stack(vectors)

    # Perform row reduction (Gaussian elimination)
    rank = np.linalg.matrix_rank(A)

    # If the rank of the matrix is less than the number of vectors, they are linearly dependent
    if rank < A.shape[1]:
        print("The vectors are linearly dependent.")
        # Find a non-trivial solution to the equation A * c = 0
        # where c is the vector of coefficients
        # Solve A * c = 0 (using least squares)
        c = np.linalg.lstsq(A, np.zeros(A.shape[0]), rcond=None)[0]
        print("Non-trivial linear combination (coefficients):")
        print(c)
    else:
        print("The vectors are linearly independent.")

def generate_linear_combination(vectors, coefficients):
    # Linear combination of vectors based on given coefficients
    result = np.zeros_like(vectors[0])
    for i, vec in enumerate(vectors):
        result += coefficients[i] * vec
    return result

# Example: 3 vectors in R^3 (3D space)
v1 = np.array([1, 2, 3])
v2 = np.array([2, 4, 6])
v3 = np.array([3, 6, 9])
```

Ln: 11 Col: 0

OUTPUT


IDLE Shell 3.13.0
File Edit Shell Debug Options Window Help
Python 3.13.0 (tags/v3.13.0:60403a5, Oct 7 2024, 09:38:07) [MSC v.1941 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/PaySprint/AppData/Local/Programs/Python/Python313/PR-05.py =
The vectors are linearly dependent.
Non-trivial linear combination (coefficients):
[0. 0. 0.]
Generated linear combination:
[3 6 9]
>>>
Ln: 11 Col: 0

PRACTICAL -06

#Check the diagonalizable property of matrices and find the corresponding eigenvalue and verify the Cayley Hamilton theorem.

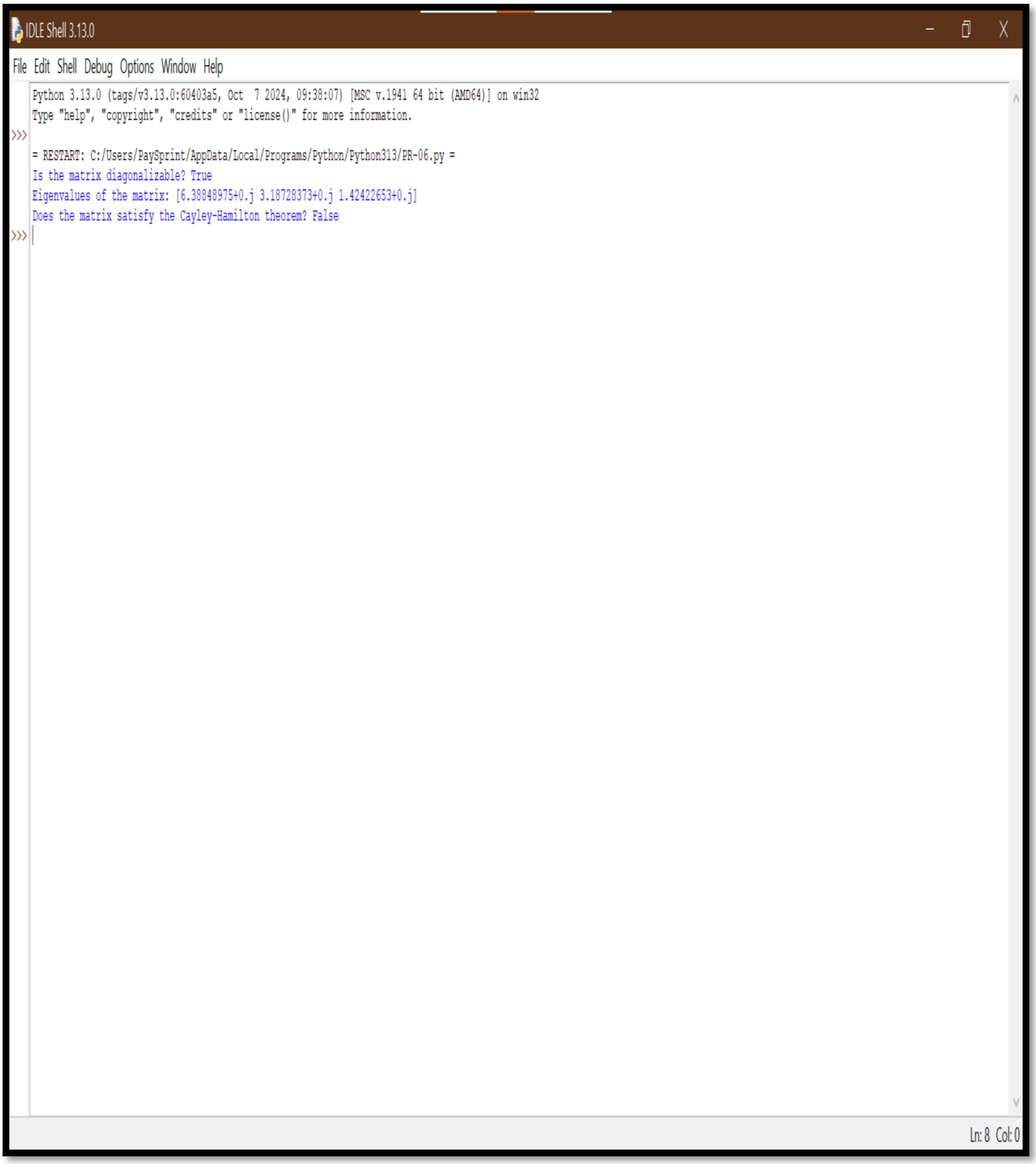
PR-06.py - C:/Users/PaySprint/AppData/Local/Programs/Python/Python313/PR-06.py (3.13.0)

File Edit Format Run Options Window Help

```python
#PRACTICAL-06
import numpy as np
from scipy.linalg import eig
from sympy import Matrix, symbols, det

# Function to check diagonalizability
def is_diagonalizable(A):
    # Compute eigenvalues and eigenvectors
    eigenvalues, eigenvectors = eig(A)

    # Check the geometric multiplicity (number of independent eigenvectors)
    # If the number of independent eigenvectors matches the size of the matrix, it's diagonalizable
    rank = np.linalg.matrix_rank(eigenvectors)
    if rank == A.shape[0]:
        return True, eigenvalues
    else:
        return False, eigenvalues

# Function to verify the Cayley-Hamilton theorem
def verify_cayley_hamilton(A):
    # Convert matrix to sympy Matrix
    A_sympy = Matrix(A)

    # Compute the characteristic polynomial
    lambda_symbol = symbols('lambda')
    char_poly = A_sympy.charpoly(lambda_symbol)

    # The characteristic polynomial as a sympy expression
    characteristic_polynomial = char_poly.as_expr()

    # Replace lambda with the matrix A in the characteristic polynomial
    A_substitution = characteristic_polynomial.subs(lambda_symbol, A_sympy)

    # Check if the result is the zero matrix (Cayley-Hamilton should hold)
    return A_substitution.is_zero

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

# Step 1: Check if the matrix is diagonalizable
diagonalizable, eigenvalues = is_diagonalizable(A)
print("Is the matrix diagonalizable?", diagonalizable)
print("Eigenvalues of the matrix:", eigenvalues)

# Step 2: Verify Cayley-Hamilton theorem
is_cayley_hamilton_true = verify_cayley_hamilton(A)
print("Does the matrix satisfy the Cayley-Hamilton theorem?", is_cayley_hamilton_true)
```

Ln: 6 Col: 0

OUTPUT


IDLE Shell 3.13.0
File Edit Shell Debug Options Window Help
Python 3.13.0 (tags/v3.13.0:60403a5, Oct 7 2024, 09:38:07) [MSC v.1941 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/PaySprint/AppData/Local/Programs/Python/Python313/PR-06.py =
Is the matrix diagonalizable? True
Eigenvalues of the matrix: [6.38848975+0.j 3.18728373+0.j 1.42422653+0.j]
Does the matrix satisfy the Cayley-Hamilton theorem? False
>>>
Ln: 8 Col: 0

PRACTICAL -07

Compute Gradient of a scalar field, Divergence and Curl of a vector filed.

```python
#PRACTICAL -07
import sympy as sp

# Define symbols (coordinates)
x, y, z = sp.symbols('x y z')

# Example Scalar Field f(x, y, z)
f = x**2 + y**2 + z**2

# Example Vector Field A(x, y, z)
A_x = x * y
A_y = y * z
A_z = z * x
A = sp.Matrix([A_x, A_y, A_z])

# 1. Compute the Gradient of the scalar field f
gradient_f = sp.Matrix([sp.diff(f, var) for var in (x, y, z)])
print("Gradient of f(x, y, z):")
sp.pprint(gradient_f)

# 2. Compute the Divergence of the vector field A
divergence_A = sp.diff(A_x, x) + sp.diff(A_y, y) + sp.diff(A_z, z)
print("\nDivergence of A(x, y, z):")
sp.pprint(divergence_A)

# 3. Compute the Curl of the vector field A
curl_A = sp.Matrix([
    sp.diff(A_z, y) - sp.diff(A_y, z),  # i-component
    sp.diff(A_x, z) - sp.diff(A_z, x),  # j-component
    sp.diff(A_y, x) - sp.diff(A_x, y)   # k-component
])
print("\nCurl of A(x, y, z):")
sp.pprint(curl_A)
```

OUTPUT

```
Python 3.13.0 (tags/v3.13.0:60403a5, Oct  7 2024, 09:38:07) [MSC v.1941 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 
= RESTART: C:/Users/PaySprint/AppData/Local/Programs/Python/Python313/PR-07.py =
Gradient of f(x, y, z):
⎡2⋅x⎤
⎢   ⎥
⎢2⋅y⎥
⎢   ⎥
⎣2⋅z⎦

Divergence of A(x, y, z):
x + y + z

Curl of A(x, y, z):
⎡-y⎤
⎢  ⎥
⎢-z⎥
⎢  ⎥
⎣-x⎦
>>> 
```