Maths Practicals.py - C:\Users\Harsh\AppData\Local\Programs\Python\Python311\Maths Practicals.py (3.11.0)

File Edit Format Run Options Window Help

```
                                    #Maths Practicals:
import numpy as np
from sympy import Matrix
from sympy import *


#1.Create and transform vectors and matrices and transpose them:

#Creating a Vector
vector = np.array([9, 8, 7])
print("Vector is:\n", vector )


# Transpose the Vector
print("\nTRANSPOSE THE VECTOR\n")

vector_transpose = np.transpose(vector)
print("Transpose of  The Given Vector is:\n", vector_transpose)
```

```
Vector is:
 [9 8 7]

TRANSPOSE THE VECTOR

Transpose of  The Given Vector is:
 [9 8 7]
```

```python
# Creating a Matrix

matrix = np.array([[6,7,9], [5,7,1], [2,0,4]])
print("Matrix is:\n",matrix)

# Transpose of Matrix
print("\nTRANSPOSE THE MATRIX\n")
matrix_transpose = np.transpose(matrix)
print("Transpose of The Given  Matrix is:\n", matrix_transpose)

# Conjugate Transpose of The Matrix

print("\n CONJUGATE TRANSPOSE THE MATRIX \n")
matrix_conjugate_transpose = np.transpose(matrix - 1j*matrix)
print("Conjugate Transpose of The Given matrix is:\n", matrix_conjugate_transpose)

print("..................................................................")
```

```
Matrix is:
 [[6 7 9]
 [5 7 1]
 [2 0 4]]

TRANSPOSE THE MATRIX

Transpose of The Given  Matrix is:
 [[6 5 2]
 [7 7 0]
 [9 1 4]]

 CONJUGATE TRANSPOSE THE MATRIX

Conjugate Transpose of The Given matrix is:
 [[6.-6.j 5.-5.j 2.-2.j]
 [7.-7.j 7.-7.j 0.+0.j]
 [9.-9.j 1.-1.j 4.-4.j]]
............................................................
```

```python
#2.Generate The Matrix into Echelon Form and Find its Rank.

# Creating a Matrix

matrix= Matrix([[3, 7, 9,],[5, 2, 1,],[-2, -9, 8]])
print("Matrix is \n: ")
print(np.array(matrix).astype(np.float64))

#Finding Reduced Row Echelon Form of The Matrix

def echelon_form(matrix):
    print("\n ROW REDUCED ECHELON FORM OF THE MATRIX\n")
    print("Reduced Row Echelon Form is \n: ")
    print(np.array(matrix.rref()[0]).astype(np.float64))
print(echelon_form(matrix))

#Finding Rank of The Matrix

def rank_matrix(matrix):
    print("\n RANK OF THE MATRIX\n")
    print("Rank of matrix is \n: ")
    print(np.linalg.matrix_rank(np.array(matrix).astype(np.float64)))
print(rank_matrix(matrix))

print("...............................................................")
```

```
Matrix is
:
[[ 3.  7.  9.]
 [ 5.  2.  1.]
 [-2. -9.  8.]]

 ROW REDUCED ECHELON FORM OF THE MATRIX

Reduced Row Echelon Form is
:
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
None

 RANK OF THE MATRIX

Rank of matrix is
:
3
None
.....................................................................
```

```python
#3.Find Cofactors, Determinant, Adjoint and Inverse of a Matrix.

# Creating a Matrix

matrix = np.array([[5,3,4], [3,1,-2], [-2,0,-3]])
print("Matrix is:\n", matrix)

#Cofactors of a Matrix:

print("\n COFACTORS OF THE MATRIX\n")
def cofactor(matrix):
    n = matrix.shape[0]
    cofactor_matrix = np.zeros(matrix.shape)
    for i in range(n):
        for j in range(n):
            sub_matrix = np.delete(np.delete(matrix, i, axis=0), j, axis=1)
            sign = (-1) ** (i + j)
            cofactor_matrix[i, j] = sign * np.linalg.det(sub_matrix)
    return cofactor_matrix
cofactor_matrix = cofactor(matrix)
print("Cofactors of The Given Matrix is:")
print(cofactor_matrix)

#Adjoint of a Matrix:

print("\n ADJOINT OF THE MATRIX\n")
def adjoint(matrix):
    return np.transpose(cofactor(matrix))
adjoint_matrix = adjoint(matrix)
print("Adjoint of The Given Matrix is:\n",adjoint_matrix)
```

```python
#Determinant of a Matrix:

def determinant(matrix):
    print("\n DETERMINANT OF THE MATRIX\n")
    determinant_matrix=np.linalg.det(matrix)
    print("Determinant of The Given Matrix is:\n",determinant_matrix)
print(determinant(matrix))

#Inverse of a Matrix:

print("\n INVERSE OF THE MATRIX\n")
inverse_matrix=np.linalg.inv(matrix)
print("Inverse of The Given Matrix is:\n",inverse_matrix)

print("............................................................................")
```

```
Matrix is:
 [[ 5  3  4]
 [ 3  1 -2]
 [-2  0 -3]]

 COFACTORS OF THE MATRIX

Cofactors of The Given Matrix is:
[[ -3.  13.   2.]
 [  9.  -7.  -6.]
 [-10.  22.  -4.]]

 ADJOINT OF THE MATRIX

Adjoint of The Given Matrix is:
 [[ -3.   9. -10.]
 [ 13.  -7.  22.]
 [  2.  -6.  -4.]]

 DETERMINANT OF THE MATRIX

Determinant of The Given Matrix is:
 32.0
None

 INVERSE OF THE MATRIX

Inverse of The Given Matrix is:
 [[-0.09375  0.28125 -0.3125 ]
 [ 0.40625 -0.21875  0.6875 ]
 [ 0.0625  -0.1875  -0.125  ]]
..........................................................................
```

```python
#4.Solve a system of Homogeneous and non-homogeneous equations using Gauss elimination method.

print("\n GAUSS ELIMINATION OF THE MATRIX\n")
def gauss_elimination(A, b):
    n = A.shape[0]
    for i in range(n):
        # Find the pivot element
        max_element = np.abs(A[i, i])
        max_row = i
        for k in range(i + 1, n):
            if np.abs(A[k, i]) > max_element:
                max_element = np.abs(A[k, i])
                max_row = k
        # Swap the current row with the pivot row
        if max_row != i:
            A[[i, max_row]] = A[[max_row, i]]
            b[[i, max_row]] = b[[max_row, i]]
        # Eliminate all elements below the pivot element
        for k in range(i + 1, n):
            c = -A[k, i] / A[i, i]
            A[k, i:] = A[k, i:] + c * A[i, i:]
            b[k] = b[k] + c * b[i]
    # Back-substitution to find the solution
    x = np.zeros(n)
    for i in range(n - 1, -1, -1):
        x[i] = (b[i] - np.dot(A[i, i + 1:], x[i + 1:])) / A[i, i]
    return x
# Example system of equations for a homogeneous case
A = np.array([[1, 2, 6], [0, 1, 4], [5, 6, 0]])
b = np.array([2, 2, 4])
x = gauss_elimination(A, b)
print("Homogeneous solution of given Equations is:")
print(x)

# Example System of Equations For a Non-Homogeneous Case
A = np.array([[1, 2, 3], [0, 1, 4], [5, 6, 0]])
b = np.array([2, 7, -20])
x = gauss_elimination(A, b)
print("Non-homogeneous solution of given Equations is:")
print(x)

print("..........................................................................")
```

```
 GAUSS ELIMINATION OF THE MATRIX

Homogeneous solution of given Equations is:
[-0.8         1.33333333  0.16666667]
Non-homogeneous solution of given Equations is:
[-2.8 -1.   2. ]
.................................................................
```

```python
#5.Solve a System of Homogeneous Equations Using The Gauss Jordan method:

print("\n GAUSS JORDAN OF THE MATRIX\n")
def gauss_jordan(A):
    n = A.shape[0]
    for i in range(n):
        # Find the pivot element
        max_element = np.abs(A[i, i])
        max_row = i
        for k in range(i + 1, n):
            if np.abs(A[k, i]) > max_element:
                max_element = np.abs(A[k, i])
                max_row = k
        # Swap the current row with the pivot row
        if max_row != i:
            A[[i, max_row]] = A[[max_row, i]]
        # Normalize the current row
        A[i, :] = A[i, :] / A[i, i]
        # Eliminate all elements above and below the pivot element
        for k in range(n):
            if k != i:
                c = A[k, i]
                A[k, :] = A[k, :] - c * A[i, :]
    return A

# Example Homogeneous Equation:
A = np.array([[1, 2, 3], [4,5, 2], [5, 7, 0]])
A = gauss_jordan(A)
print("Reduced row-echelon form of The Given Matrix is:\n",A)

print("..................................................................")
```

GAUSS JORDAN OF THE MATRIX

```
Reduced row-echelon form of The Given Matrix is:
 [[1 0 0]
 [0 1 0]
 [0 0 1]]
.........................................................................
```

```
#6.Generate Basis of Column Space, Null Space, Row Space and Left Null Space of a Matrix Space

matrix = Matrix([[8,2,4], [3,6,2], [9,7,1]])
print("Matrix is:\n", matrix)

#To Find The Column Space of The Matrix:

print("\n COLUMN SPACE OF THE MATRIX\n")
matrix_columnspace=matrix.columnspace()
print("\nThe Column Space of the Matrix is:\n",matrix_columnspace)

#To Find The Row Space of The Matrix:

print("\n ROW SPACE OF THE MATRIX\n")
matrix_rowspace=matrix.rowspace()
print("\nThe Row Space of the Matrix is:\n",matrix_rowspace)

#To Find The Null Space of The Matrix:

print("\n NULL SPACE OF THE MATRIX\n")
matrix_nullspace=matrix.nullspace()
print("\nThe Null Space of the Matrix is:\n",matrix_nullspace)


#To Find The Left Null Space of The Matrix:

print("\n LEFT NULL SPACE OF THE MATRIX\n")
AB = matrix.T
matrix_leftnullspace = AB.nullspace()

print("\nMatrix Transpose is :\n ")
print(AB)

print("\nLeft Null Space of the Matrix is: \n")
print(matrix_leftnullspace)

print("..........................................................................")
```

```
Matrix is:
 Matrix([[8, 2, 4], [3, 6, 2], [9, 7, 1]])

 COLUMN SPACE OF THE MATRIX


The Column Space of the Matrix is:
 [Matrix([
[8],
[3],
[9]]), Matrix([
[2],
[6],
[7]]), Matrix([
[4],
[2],
[1]])]

 ROW SPACE OF THE MATRIX


The Row Space of the Matrix is:
 [Matrix([[8, 2, 4]]), Matrix([[0, 42, 4]]), Matrix([[0, 0, -1328]])]

 NULL SPACE OF THE MATRIX


The Null Space of the Matrix is:
 []

 LEFT NULL SPACE OF THE MATRIX


Matrix Transpose is :

Matrix([[8, 3, 9], [2, 6, 7], [4, 2, 1]])

Left Null Space of the Matrix is:

[]
.....................................................................
```

```python
#10.>Application of Linear algebra: Coding and decoding of messages using nonsingular

#ENCODE  OF THE MATRIX

print("\n ENCODE  OF THE MATRIX\n")
string="harsh"
hoo=string.encode(encoding='utf-8')
print("The Encoded Version Of The String IS: \n",hoo)

#DECODE  OF THE MATRIX
print("\n DECODE  OF THE MATRIX\n")
harp=hoo.decode()
print("The Decoded Version/Original String IS: \n",harp)

print(".....................................................")
```

```
 ENCODE  OF THE MATRIX

The Encoded Version Of The String IS:
 b'harsh'

 DECODE  OF THE MATRIX

The Decoded Version/Original String IS:
 harsh
```

```python
#11.>Compute Gradient of a scalar field.
print("\n GRADIENT FIELD OF THE MATRIX\n")
matrix=np.array([2, 8, 7, 9, 69, 43],dtype=float)

print("Original Matrix  : \n", matrix)
gradient_matrix=np.gradient(matrix)
print("Gradient Of The Given Matrix is : \n",gradient_matrix)

print("............................................................................")
```

```
 GRADIENT FIELD OF THE MATRIX

Original Matrix  :
 [ 2.  8.  7.  9. 69. 43.]
Gradient Of The Given Matrix is :
 [  6.    2.5   0.5  31.   17.  -26. ]
...............................................................................
```

```
#12.>Compute Divergence of a vector field

print("\n DIVERGENCE  OF THE MATRIX\n")
#Creating a Matrix

matrix= np.array([6, 4, 9, 13, 22, 69], dtype=float)

#Finding Divergence of a Matrix

def divergence(F):
    return np.ufunc.reduce(np.add,np.gradient(F))

print("Input  :\n ", matrix)

print("Divergence :\n ",divergence(matrix))

print("............................................................................")
```

```
 DIVERGENCE  OF THE MATRIX

Input  :
  [ 6.  4.  9. 13. 22. 69.]
Divergence :
  85.5
............................................................................
```

```python
#13.>Compute Curl of a vector field.

print("\n CURL OF THE VECTOR FIELD\n")

from sympy.physics.vector import ReferenceFrame
from sympy.physics.vector import curl
R= ReferenceFrame('R')

F= R[1]**2 * R[2] * R.x - R[0]*R[1] * R.y + R[2]**2 * R.z

print("\nVector is: ", F)

curl = curl(F, R)
print("\nCurl of THE Given Vector is : ",curl)


print("..................................................................")
```

```
 CURL OF THE VECTOR FIELD


Vector is:  R_y**2*R_z*R.x - R_x*R_y*R.y + R_z**2*R.z

Curl of THE Given Vector is :  R_y**2*R.y + (-2*R_y*R_z - R_y)*R.z
.....................................................................
```