LAB RECORD

Name: Jeevan Koshy

Roll No: 1740256

Revision

AIM ~ To get familiar with the environment of Jupyter using logical questions

1. WAP to convert the month name to the number of days of that particular month.

In [1]:

```
def mon(month):
    if(month=="Jan" or "January"):
        print("31")
    elif(month=="February" or month=="Feb"):
        print("28")
    elif(month=="March" or month=="Mar"):
        print("31")
    elif(month=="April" or month=="Apr"):
        print("30")
    elif(month=="May"):
        print("31")
    elif(month=="June" or month=="Jun"):
        print("30")
    elif(month=="July" or month=="Jul"):
        print("31")
    elif(month=="August" or month=="Aug"):
        print("31")
    elif(month=="September" or month=="Sep"):
        print("30")
    elif(month=="October" or month=="Oct"):
        print("31")
    elif(month=="November" or month=="Nov"):
        print("30")
    elif(month=="December" or month=="Dec"):
        print("31")

month = input("Enter any month: ")
mon(month)
```

```
Enter any month: Oct
31
```

2. WAP to display the grades of a student based on %

In [3]:

```
def perc(x):
    if(x>90):
        print("Grade A")
    elif(x>80 and x<90):
        print("Grade B")
    elif(x>70 and x<80):
        print("Grade C")
    elif(x>60 and x<70):
        print("Grade D")
    elif(x>50 and x<60):
        print("Grade E")
    elif(x>40 and x<50):
        print("Grade F")
    else:
        print("FAIL!")

perc(76)
```

```
Grade C
```

3. WAP to take 3 sides of a triangle and display if it is scalene, iscosceles or

equilateral

In [4]:

```
print("Enter 3 sides of a triangle: ")
a = input('Enter 1st side: ')
b = input('Enter 2nd side: ')
c = input('Enter 3rd side: ')
if(a!=b and b!=c and a!=c):
    print('It is a scalene triangle!')
elif(a==b or b==c or a==c):
    print('It is an iscosceles triangle')
elif(a==b and b==c and a==c):
    print('It is an equilateral triangle')
```

```
Enter 3 sides of a triangle:
Enter 1st side: 10
Enter 2nd side: 20
Enter 3rd side: 10
It is an iscosceles triangle
```

Conclusion ~ From the above code, we found

- How to convert the name of a month to it's corresponding number of days

- To display the grades of a student based on %.

- To find out whether a triangle is scalene, iscosceles or equilateral

MATRICES

Taking input from user for matrix

In [2]:

```
def inp(m,n):
    print()
    print('There are {0} rows and {1} columns'.format(m,n))
    # Initializing matrix
    matrix = []
    print("Enter the entries rowwise:")

    # For user input
    for i in range(int(m)): # A for loop for row entries
        a =[]
        for j in range(int(n)): # A for loop for column entries
            a.append(int(input()))
        matrix.append(a)

    # For printing the matrix
    print('---------------')
    print("THE MATRIX IS: ")
    print('---------------')
    for i in range(int(m)):
        print()
        for j in range(int(n)):
            print(matrix[i][j], end = " ")
        print()


a = input('Enter the number of rows: ')
b = input('Enter the number of columns: ')
inp(a,b)
```

```
Enter the number of rows: 3
Enter the number of columns: 3

There are 3 rows and 3 columns
Enter the entries rowwise:
1
2
3
4
5
6
7
8
9
---------------
THE MATRIX IS:
---------------

1 2 3

4 5 6

7 8 9
```

AIM ~ To perform operations on Matrices with the help of in - built functions in Python

1. Define 2 matrices

In [1]:

```
import numpy as np
A = np.array([[1,2,3],[4,5,6],[7,8,9]])
B = np.array([[9,8,7],[6,5,4],[3,2,1]])
print("1st matrix = ")
print(A)
print("2nd matrix = ")
print(B)
```

```
1st matrix =
[[1 2 3]
 [4 5 6]
 [7 8 9]]
2nd matrix =
[[9 8 7]
 [6 5 4]
 [3 2 1]]
```

2. Find the order of the matrix.

In [15]:

```
ord_A = A.shape
print("The order of the 1st matrix is: ")
print(ord_A)
```

```
The order of the 1st matrix is:
(3, 3)
```

3. Perform all basic operations on matrices (+,-,*,/) & find the transpose of a matrix

Addition

In [16]:

```
A_add_B = np.add(A,B)
print("The sum of the 2 matrices are: ")
print(A_add_B)
```

```
The sum of the 2 matrices are:
[[10 10 10]
 [10 10 10]
 [10 10 10]]
```

Subtraction

In [17]:

```
A_sub_B = np.subtract(A,B)
print("The difference of the 2nd matrix from the 1st matrix is: ")
print(A_sub_B)
```

```
The difference of the 2nd matrix from the 1st matrix is:
[[-8 -6 -4]
 [-2  0  2]
 [ 4  6  8]]
```

In [18]:

```
B_sub_A = np.subtract(B,A)
print("The difference of the 1st matrix from the 2nd matrix is:")
print(B_sub_A)
```

```
The difference of the 1st matrix from the 2nd matrix is:
[[ 8  6  4]
 [ 2  0 -2]
 [-4 -6 -8]]
```

Multiplication

In [20]:

```
A_mul_B = np.dot(A,B)
print("The dot product of the 1st and 2nd matrix is: ")
print(A_mul_B)
```

```
The dot product of the 1st and 2nd matrix is:
[[ 30  24  18]
 [ 84  69  54]
 [138 114  90]]
```

In [21]:

```
B_mul_A = np.dot(B,A)
print("The dot product of the 1st and 2nd matrix is: ")
print(B_mul_A)
```

```
The dot product of the 1st and 2nd matrix is:
[[ 90 114 138]
 [ 54  69  84]
 [ 18  24  30]]
```

Division

In [24]:

```
A_div_B = np.divide(A,B)
print("The division of the 2nd matrix from the 1st matrix is: ")
print(A_div_B)
```

```
The division of the 2nd matrix from the 1st matrix is:
[[ 0.11111111  0.25        0.42857143]
 [ 0.66666667  1.          1.5       ]
 [ 2.33333333  4.          9.        ]]
```

In [25]:

```
B_div_A = np.divide(B,A)
print("The division of the 1st matrix from the 2nd matrix is: ")
print(B_div_A)
```

```
The division of the 1st matrix from the 2nd matrix is:
[[ 9.          4.          2.33333333]
 [ 1.5         1.          0.66666667]
 [ 0.42857143  0.25        0.11111111]]
```

Transpose

In [26]:

```
A_trans = np.transpose(A)
print("The transpose of the 1st matrix is: ")
print(A_trans)
```

```
The transpose of the 1st matrix is:
[[1 4 7]
 [2 5 8]
 [3 6 9]]
```

In [27]:

```
B_trans = np.transpose(B)
print("The transpose of the 2nd matrix is: ")
print(B_trans)
```

```
The transpose of the 2nd matrix is:
[[9 6 3]
 [8 5 2]
 [7 4 1]]
```

4. Find the upper and lower triangular part of a matrix

Upper triangular

In [28]:

```
A_upp = np.triu(A,k=0)
print("The upper triangular part of the matrix is: ")
print(A_upp)
```

```
The upper triangular part of the matrix is:
[[1 2 3]
 [0 5 6]
 [0 0 9]]
```

Lower triangular

In [34]:

```
A_low = np.tril(A,k=0)
print("The lower triangular part of the matrix is: ")
print(A_low)
```

```
The lower triangular part of the matrix is:
[[1 0 0]
 [4 5 0]
 [7 8 9]]
```

5. Enter a 6X6 matrix 'X' and get -

In [4]:

```
X = np.array([[1,2,3,4,5,6],[7,8,9,10,11,12],[13,14,15,16,17,18],[19,20,21,22,23,24],[25,26
print("The 6 by 6 matrix is: ")
print(X)
```

```
The 6 by 6 matrix is:
[[ 1  2  3  4  5  6]
 [ 7  8  9 10 11 12]
 [13 14 15 16 17 18]
 [19 20 21 22 23 24]
 [25 26 27 28 29 30]
 [31 32 33 34 35 36]]
```

A. Any 5 elements of the matrix 'X'

In [39]:

```
print("The 5 elements are: ")
print(X[0,0:5])
```

```
The 5 elements are:
[1 2 3 4 5]
```

B.1st, 3rd and 6th row of the matrix.

In [41]:

```
print("The 1st, 3rd and 6th row of the matrix is: ")
print(X[[0,2,5]])
```

```
The 1st, 3rd and 6th row of the matrix is:
[[ 1  2  3  4  5  6]
 [13 14 15 16 17 18]
 [31 32 33 34 35 36]]
```

C. 3rd to 6th row of the matrix

In [42]:

```
print("The 3rd to 6th row of the matrix is: ")
print(X[2:])
```

```
The 3rd to 6th row of the matrix is: 
[[13 14 15 16 17 18]
 [19 20 21 22 23 24]
 [25 26 27 28 29 30]
 [31 32 33 34 35 36]]
```

D. 4th and 6th row of the matrix

In [43]:

```
print("The 4th and 6th row of the matrix is" )
print(X[[3,5]])
```

```
The 4th and 6th row of the matrix is
[[19 20 21 22 23 24]
 [31 32 33 34 35 36]]
```

E. 1st row and all the columns of the matrix

In [44]:

```
print("The 1st row and all the columns of the matrix are:")
print(X[0,:])
```

```
The 1st row and all the columns of the matrix are:
[1 2 3 4 5 6]
```

F. All the rows and 3rd to 6th column of the matrix

In [47]:

```
print("All the rows from the 3rd to 6th column of the matrix are: ")
print(X[:,2:6])
```

```
All the rows from the 3rd to 6th column of the matrix are: 
[[ 3  4  5  6]
 [ 9 10 11 12]
 [15 16 17 18]
 [21 22 23 24]
 [27 28 29 30]
 [33 34 35 36]]
```

G. All the rows and second column of the matrix

In [5]:

```
print("The 2nd column of the matrix with all of its rows are: ")
print(X[:,[1]])
```

```
The 2nd column of the matrix with all of its rows are:
[[ 2]
 [ 8]
 [14]
 [20]
 [26]
 [32]]
```

H. 1st, 2nd, 5th column of the matrix

In [6]:

```
print("The 1st, 2nd and 5th column of the matrix are - ")
print(X[:,[0,1,4]])
```

```
The 1st, 2nd and 5th column of the matrix are -
[[ 1  2  5]
 [ 7  8 11]
 [13 14 17]
 [19 20 23]
 [25 26 29]
 [31 32 35]]
```

I. Square root of each element of the matrix

In [7]:

```
print("The square root of each element of the matrix is: ")
print(np.sqrt(X))
```

```
The square root of each element of the matrix is:
[[ 1.          1.41421356  1.73205081  2.          2.23606798  2.44948974]
 [ 2.64575131  2.82842712  3.          3.16227766  3.31662479  3.46410162]
 [ 3.60555128  3.74165739  3.87298335  4.          4.12310563  4.24264069]
 [ 4.35889894  4.47213595  4.58257569  4.69041576  4.79583152  4.89897949]
 [ 5.          5.09901951  5.19615242  5.29150262  5.38516481  5.47722558]
 [ 5.56776436  5.65685425  5.74456265  5.83095189  5.91607978  6.        ]]
```

J. Row wise summation ,column wise summation

In [10]:

```
print("Summation of 1st row: ")
print(np.sum(X[0,0:]))
print("Summation of 2nd row: ")
print(np.sum(X[1,0:]))
print("Summation of 3rd row: ")
print(np.sum(X[2,0:]))
print("Summation of 4th row: ")
print(np.sum(X[3,0:]))
print("Summation of 5th row: ")
print(np.sum(X[4,0:]))
print("Summation of 6th row: ")
print(np.sum(X[5,0:]))
```

```
Summation of 1st row:
21
Summation of 2nd row:
57
Summation of 3rd row:
93
Summation of 4th row:
129
Summation of 5th row:
165
Summation of 6th row:
201
```

K. Sum of all the elements.

In [11]:

```
print("The sum of all the elements in the matrix is: ")
np.sum(X[:,:])
```

```
The sum of all the elements in the matrix is:
```

Out[11]:

```
666
```

6. Solve the following system of linear equation using built in functions.

A.

x+y-2z = 1

2x-3y+z = -8

3x+y+4z = 7

In [12]:

```
a = np.array([[1,1,-2],[2,-3,1],[3,1,4]])
b = np.array([1,-8,7])
print(np.linalg.solve(a,b))
```

```
[ -2.96059473e-16   3.00000000e+00   1.00000000e+00]
```

B.

x+2y-z = 1

2x+y+4z = 2

3x+3y+4z = 1

In [13]:

```
a = np.array([[1,2,-1],[2,1,4],[3,3,4]])
b = np.array([1,2,1])
print(np.linalg.solve(a,b))
```

```
[ 7. -4. -2.]
```

C.

x-y+z-s = 2

x-y+z+s = 0

4x-4y+4z = 4

-2x+2y-2z+s = -3

In [14]:

```
a = np.array([[1,-1,1,-1],[1,-1,1,1],[4,-4,4],[-2,2,-2,1]])
b = np.array([2,0,4,-3])
np.linalg.solve(a,b)
# The matrix is singular and the determinant is 0.
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
<ipython-input-14-dda4fb9f66d2> in <module>()
      1 a = np.array([[1,-1,1,-1],[1,-1,1,1],[4,-4,4],[-2,2,-2,1]])
      2 b = np.array([2,0,4,-3])
----> 3 np.linalg.solve(a,b)
      4 # The matrix is singular and the determinant is 0.

C:\Users\Jeevan\Anaconda3\lib\site-packages\numpy\linalg\linalg.py in solve
(a, b)
    355     """
    356     a, _ = _makearray(a)
--> 357     _assertRankAtLeast2(a)
    358     _assertNdSquareness(a)
    359     b, wrap = _makearray(b)

C:\Users\Jeevan\Anaconda3\lib\site-packages\numpy\linalg\linalg.py in _asser
tRankAtLeast2(*arrays)
    200         if len(a.shape) < 2:
    201             raise LinAlgError('%d-dimensional array given. Array mus
t be '
--> 202                     'at least two-dimensional' % len(a.shape))
    203 
    204 def _assertSquareness(*arrays):

LinAlgError: 1-dimensional array given. Array must be at least two-dimension
al
```

D.

2x+y=3

4x+2y=6

In [15]:

```
a = np.array([[2,1],[4,2]])
b = np.array([3,6])
np.linalg.solve(a,b)
# The matrix form is singular and hence cannot be solved
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
<ipython-input-15-5eb615042530> in <module>()
      1 a = np.array([[2,1],[4,2]])
      2 b = np.array([3,6])
----> 3 np.linalg.solve(a,b)
      4 # The matrix form is singular and hence cannot be solved

C:\Users\Jeevan\Anaconda3\lib\site-packages\numpy\linalg\linalg.py in solve
(a, b)
    382     signature = 'DD->D' if isComplexType(t) else 'dd->d'
    383     extobj = get_linalg_error_extobj(_raise_linalgerror_singular)
--> 384     r = gufunc(a, b, signature=signature, extobj=extobj)
    385 
    386     return wrap(r.astype(result_t, copy=False))

C:\Users\Jeevan\Anaconda3\lib\site-packages\numpy\linalg\linalg.py in _raise
_linalgerror_singular(err, flag)
     88 
     89 def _raise_linalgerror_singular(err, flag):
---> 90     raise LinAlgError("Singular matrix")
     91 
     92 def _raise_linalgerror_nonposdef(err, flag):

LinAlgError: Singular matrix
```

E.

x+2y-z = 1

2x+y+5z = 2

3x+3y+4z = 1

In [16]:

```
a = np.array([[1,2,-1],[2,1,5],[3,3,4]])
b = np.array([1,2,1])
print(np.linalg.solve(a,b))
```

```
[ -1.04293886e+16   6.63688366e+15   2.84437871e+15]
```

F.

2x+y-2z = -3

x-3y+z = 8

4x-y-2z = 3

In [17]:

```
a = np.array([[2,1,-2],[1,-3,1],[4,-1,-2]])
b = np.array([-3,8,3])
print(np.linalg.solve(a,b))
```

```
[ 2. -1.  3.]
```

G.

2x+y=3

2x-y=0

x-2y=4

In [19]:

```
a = np.array([[2,1],[2,-1],[1,-2]])
b = np.array([[3,0,4]])
np.linalg.solve(a,b)
# Since the matrix is square matrix it cannot be solved
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
<ipython-input-19-7460f17ef1cf> in <module>()
      1 a = np.array([[2,1],[2,-1],[1,-2]])
      2 b = np.array([[3,0,4]])
----> 3 np.linalg.solve(a,b)
      4 # Since the matrix is square matrix it cannot be solved

C:\Users\Jeevan\Anaconda3\lib\site-packages\numpy\linalg\linalg.py in solve
(a, b)
    356     a, _ = _makearray(a)
    357     _assertRankAtLeast2(a)
--> 358     _assertNdSquareness(a)
    359     b, wrap = _makearray(b)
    360     t, result_t = _commonType(a, b)

C:\Users\Jeevan\Anaconda3\lib\site-packages\numpy\linalg\linalg.py in _asser
tNdSquareness(*arrays)
    210     for a in arrays:
    211         if max(a.shape[-2:]) != min(a.shape[-2:]):
--> 212             raise LinAlgError('Last 2 dimensions of the array must b
e square')
    213 
    214 def _assertFinite(*arrays):

LinAlgError: Last 2 dimensions of the array must be square
```

H.

2x+y-2z = 10

3x+2y+2z = 1

5x+4y+3z = 4

In [18]:

```
a = np.array([[2,1,-2],[3,2,2],[5,4,3]])
b = np.array([10,1,4])
print(np.linalg.solve(a,b))
```

```
[ 1.  2. -3.]
```

I.

x+2y-3z = -1

3x-y+2z = 7

5x+3y-4z = 2

In [21]:

```
a = np.array([[1,2,-3],[3,-1,2],[5,3,-4]])
b = np.array([-1,7,2])
print(np.linalg.solve(a,b))
```

```
[  8.77324603e+14  -9.65057063e+15  -6.14127222e+15]
```

Conclusion ~ Different operations on Matrices have been performed such as -

- Finding the order of a matrix

- Performing all the basic operations on matrices (+,-,*,/)

- Finding the transpose of a matrix

- Finding the upper and lower triangular parts of a matrix

- Extracting different elements from a matrix

- Finding the square root of each element in a matrix

- Finding row wise summation and column wise summation of a matrix

- Finding the sum of all elements in a matrix

- Solving a system of Linear Equations

EIGEN VALUES AND VECTORS

In [2]:

```
import numpy as np
A = np.mat(input('Enter elements: '))
print(A)
```

```
Enter elements: 1,2,3;4,5,6
[[1 2 3]
 [4 5 6]]
```

In [1]:

```
import numpy as np
A = np.array([[-1,2],[-6,6]])
print(np.linalg.eig(A))
```

```
(array([ 2.,  3.]), array([[-0.5547002 , -0.4472136 ],
       [-0.83205029, -0.89442719]]))
```

Applying eigen value properties on a matrix

In [3]:

```
def prop():
    print("The eigen value properties of a matrix are: ")
    print("_____________________________________________________")
    print("(i)      λ(1) + λ(2) + λ(3) + ................ λ(n) = sum(trace(A))")
    print("(ii)     λ(1) * λ(2) * λ(3) * ................ λ(n) = det(A)")
    print("(iii)    A = λ; A^T = λ")
    print("(iv)     A = λ; A^-1 = λ^-1")
    print("(v)      Eigen values of an identity matrix is 1")
    print("(vi)     Idempotent matrix A^2 = A, λ = 0,1 ")
    print("(vii)    Skew symmetric matrix(A^T = -A), λ either be 0 or purely imaginary ")
    print("(viiii)  Orthogonal matrix A^T = A^-1; mod. λ = 1; λ = 1,-1 ")
    print("(ix)     For upper/lower triangular matrix, λ = diag(A) {diagonal elements} ")
    print("(x)      For a real symmetric matrix, λ is always is always real  ")
    print("(xi)     Hermitian Matrix ")

prop()

import numpy as np

def inp(m,n):
    print()
    print('There are {0} rows and {1} columns'.format(m,n))
    # Initializing matrix
    matrix = []
    tr = []
    mat_trans = []
    print("Enter the entries rowwise:")

    # For user input
    for i in range(int(m)): # A for loop for row entries
        a = []
        for j in range(int(n)): # A for loop for column entries
            a.append(int(input()))
        matrix.append(a)

    # For printing the matrix
    print('---------------')
    print("THE MATRIX IS: ")
    print('---------------')
    for i in range(int(m)):
        print()
        for j in range(int(n)):
            print(matrix[i][j], end = " ")
            if(i==j):
                tr.append(matrix[i][j])
        print()

    # For finding transpose
    for i in range(int(m)):
        b = []
        for j in range(int(n)):
            b.append(matrix[j][i])
        mat_trans.append(b)


    print('----------------------------------------------------------------')
    print('(i)')
    print("The trace of the matrix is: {}".format(sum(tr)))
    print("The sum of the eigen values of the matrix is: {}".format(sum(np.linalg.eigvals(m
```

```
    print("Therefore the sum of trace of the matrix is equal to the sum of the diagonal ele
    print('----------------------------------------------------------------')
    print('(ii)')
    print("The product of the eigen values of the matrix is: {}".format(np.prod(np.linalg.e
    print('The determinant of the matrix is: {}'.format(np.linalg.det(matrix)))
    print('Therefore the determinant of the matrix is equal to the product of the eigen val
    print('----------------------------------------------------------------')
    print('The transpose of the matrix is: ')
    print('----------------------------------------------------------------')
    for i in range(int(m)):
        print()
        for j in range(int(n)):
            print(mat_trans[i][j], end = " ")
        print()
    print('----------------------------------------------------------------')
    print('(iv)')
    print('The eigen values of the transpose matrix are: ')
    print(np.linalg.eigvals(mat_trans))
    print('Therefore the eigen values of the matrix are equal to the eigen values of the tr
    print('----------------------------------------------------------------')
    print('(v)')
    print('The eigen values of the matrix are: ')
    print(np.linalg.eigvals(matrix))
    print('Therefore the Eigen values of an identity matrix is 1')
    print('----------------------------------------------------------------')
    print('(vi)')
    print('The square of the matrix is: \n{}'.format(np.square(matrix)))
    print('Therefore the square of the matrix is equal to the matrix itself and it is an Id
    print('----------------------------------------------------------------')

print('----------------------------------------------------------------')
a = input('Enter the number of rows: ')
b = input('Enter the number of columns: ')
inp(a,b)
```

```
The eigen value properties of a matrix are:
_____________________________________________________
(i)      λ(1) + λ(2) + λ(3) + ................ λ(n) = sum(trace(A))
(ii)     λ(1) * λ(2) * λ(3) * ................ λ(n) = det(A)
(iii)    A = λ; A^T = λ
(iv)     A = λ; A^-1 = λ^-1
(v)      Eigen values of an identity matrix is 1
(vi)     Idempotent matrix A^2 = A, λ = 0,1
(vii)    Skew symmetric matrix(A^T = -A), λ either be 0 or purely imaginary
(viiii)  Orthogonal matrix A^T = A^-1; mod. λ = 1; λ = 1,-1
(ix)     For upper/lower triangular matrix, λ = diag(A) {diagonal elements}
(x)      For a real symmetric matrix, λ is always is always real
(xi)     Hermitian Matrix
----------------------------------------------------------------
Enter the number of rows: 3
Enter the number of columns: 3

There are 3 rows and 3 columns
Enter the entries rowwise:
1
0
0
0
1
0
```

```
0
0
1
---------------
THE MATRIX IS:
---------------

1 0 0

0 1 0

0 0 1
-----------------------------------------------------------------
(i)
The trace of the matrix is: 3
The sum of the eigen values of the matrix is: 3.0
Therefore the sum of trace of the matrix is equal to the sum of the diagonal
elements in a matrix
-----------------------------------------------------------------
(ii)
The product of the eigen values of the matrix is: 1.0
The determinant of the matrix is: 1.0
Therefore the determinant of the matrix is equal to the product of the eigen
values of the matrix
-----------------------------------------------------------------
The transpose of the matrix is:
-----------------------------------------------------------------

1 0 0

0 1 0

0 0 1
-----------------------------------------------------------------
(iv)
The eigen values of the transpose matrix are:
[ 1.  1.  1.]
Therefore the eigen values of the matrix are equal to the eigen values of th
e transpose matrix
-----------------------------------------------------------------
(v)
The eigen values of the matrix are:
[ 1.  1.  1.]
Therefore the Eigen values of an identity matrix is 1
-----------------------------------------------------------------
(vi)
The square of the matrix is:
[[1 0 0]
 [0 1 0]
 [0 0 1]]
Therefore the square of the matrix is equal to the matrix itself and it is a
n Idempotent matrix which has eigen values 1
-----------------------------------------------------------------
```

From this, the eigen value properties of a matrix are proved

Diagonalise the given matrix

In [15]:

```
### import numpy as np

def diagonal():
    print("Program to diagonalise an input matrix")
    print("-------------------------------------")
    m = input('Enter the number of rows: ')
    n = input('Enter the number of columns: ')
    print('There are {0} rows and {1} columns'.format(m,n))
    # Initializing matrix
    matrix = []
    tr = []
    d = []
    print("Enter the entries rowwise:")

    # For user input
    for i in range(int(m)): # A for loop for row entries
        a = []
        for j in range(int(n)): # A for loop for column entries
            a.append(int(input()))
        matrix.append(a)

    # For printing the matrix
    print('---------------')
    print("THE MATRIX IS: ")
    print('---------------')
    for i in range(int(m)):
        print()
        for j in range(int(n)):
            print(matrix[i][j], end = " ")
            if(i==j):
                tr.append(matrix[i][j])
        print()

    print("-------------------------------------")
    ev, evec = np.linalg.eig(matrix)
    print("The eigen values are \n{}".format(np.around(ev)))
    print("-------------------------------------")
    print("The eigen vectors are \n{}".format(np.around(evec)))
    print("-------------------------------------")
    P = evec
    d = np.around(np.dot(np.linalg.inv(P),np.dot(matrix,P)))
    d = np.where(d==-0,0,d)
    print("The diagonalised matrix is: \n{}".format(np.around(d)))
    print("-------------------------------------")
    print("Checking whether both diagonalised matrices are equal")
    print("-------------------------------------")
    Diag = np.around(np.diag(ev))
    print(Diag)
    print("-------------------------------------")
    if(np.array_equal(d,Diag)):
        print("It is a diagonalised matrix")
    else:
        print("It is not a diagonalised matrix")

diagonal()
```

```
Program to diagonalise an input matrix
-------------------------------------
Enter the number of rows: 3
```

```
Enter the number of columns: 3
There are 3 rows and 3 columns
Enter the entries rowwise:
1
2
3
4
5
6
7
8
9
---------------
THE MATRIX IS:
---------------

1 2 3

4 5 6

7 8 9
--------------------------------------
The eigen values are
[ 16.  -1.  -0.]
--------------------------------------
The eigen vectors are
[[-0. -1.  0.]
 [-1. -0. -1.]
 [-1.  1.  0.]]
--------------------------------------
The diagonalised matrix is:
[[ 16.   0.   0.]
 [  0.  -1.   0.]
 [  0.   0.   0.]]
--------------------------------------
Checking whether both diagonalised matrices are equal
--------------------------------------
[[ 16.   0.   0.]
 [  0.  -1.   0.]
 [  0.   0.  -0.]]
--------------------------------------
It is a diagonalised matrix
```

Conclusion ~ Thus Caley Hamilton Theorem is proved since both the diagonalised matrices are equal

Caley Hamilton Theorem

In [19]:

```
import numpy as np

def diag():
    print("Program to diagonalise an input matrix")
    print("-------------------------------------")
    m = input('Enter the number of rows: ')
    n = input('Enter the number of columns: ')
    print('There are {0} rows and {1} columns'.format(m,n))
    # Initializing matrix
    matrix = []
    tr = []
    d = []
    print("Enter the entries rowwise:")

    # For user input
    for i in range(int(m)): # A for loop for row entries
        a = []
        for j in range(int(n)): # A for loop for column entries
            a.append(int(input()))
        matrix.append(a)

    # For printing the matrix
    print('---------------')
    print("THE MATRIX IS: ")
    print('---------------')
    for i in range(int(m)):
        print()
        for j in range(int(n)):
            print(matrix[i][j], end = " ")
            if(i==j):
                tr.append(matrix[i][j])
        print()

    sum = np.zeros((3,3))
    print("-------------------------------------")
    print("The characterisic eqn of the matrix is")
    print("-------------------------------------")
    ce_matrix = np.poly(matrix)
    cheq_matrix = np.poly1d(ce_matrix)
    print("-------------------------------------")
    print("The characterisic polynomial of the matrix is:")
    print("-------------------------------------")
    print(cheq_matrix)
    for i in range(0,len(cheq_matrix)):
        sum=sum+round(cheq_matrix[len(cheq_matrix)-(len(cheq_matrix)+(i+1))])*(matrix**i)
    print("Since the sum of the above statement is zero matrix, thus caley hamilton theorem

diag()
```

```
Program to diagonalise an input matrix
-------------------------------------
Enter the number of rows: 3
Enter the number of columns: 3
There are 3 rows and 3 columns
Enter the entries rowwise:
1
2
3
4
```

```
5
6
7
8
9
---------------
THE MATRIX IS:
---------------

1 2 3

4 5 6

7 8 9
--------------------------------------
The characterisic eqn of the matrix is
--------------------------------------
--------------------------------------
The characterisic polynomial of the matrix is:
--------------------------------------
   3      2
1 x - 15 x - 18 x - 1.757e-14

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-19-2c76112cc2dc> in <module>()
     46     print("Since the sum of the above statement is zero matrix, thus
caley hamilton theorem is verified",sum)
     47
---> 48 diag()

<ipython-input-19-2c76112cc2dc> in diag()
     43     print(cheq_matrix)
     44     for i in range(0,len(cheq_matrix)):
---> 45         sum=sum+round(cheq_matrix[len(cheq_matrix)-(len(cheq_matrix)
+(i+1))])*(matrix**i)
     46     print("Since the sum of the above statement is zero matrix, thus
caley hamilton theorem is verified",sum)
     47

TypeError: unsupported operand type(s) for ** or pow(): 'list' and 'int'
```

Thus, we can find the characteristic eqn of a matrix

Revision

1) WAP to check if the entered no. is prime or not (range 1 to 100)

2) Solve the system of linear equations and verify it using Cramers rule:

x - y - 2z = 5

x - 2y + z = -2

-2x + y + z = 4

3) WAP to check if the entered matrix is skew symmetric.

4) Find eigen values and eigen vectors of ~

[6, -2, 2]

[2, 3, -1]

[2, -1, 3]

Print the corresponding eigen vector with it's eigen values.

5) If A ~

[1 2 9 12]

[4 4 8 14]

[8 6 7 15]

[9 7 13 18]

Extract ~

a)

[1 2]

[8 6]

b)

[4 4]

[8 6]

[9 7]

c)

[2 12]

[6 15]

d)

[8 14]

[7 15]

e)

[9 12]

[8 14]

[7 15]

[13 18]

1

In [21]:

```
def prime(n):
    print("------------------------------------")
    print("The number entered is {}".format(n))
    print("------------------------------------")
    for i in range(2,n):
        if(n%i == 0):
            print("{} is not a prime number".format(n))
            break
        else:
            print("{} is a prime number".format(n))
            break

a = input("Enter a prime number: ")
prime(int(a))
```

```
Enter a prime number: 11
------------------------------------
The number entered is 11
------------------------------------
11 is a prime number
```

Conclusion ~ Thus proved that it is possible to check whether a number is prime or not

2

In [7]:

```
# 1,-1,-2;1,-2,1;-2,1,1
A = np.mat(input("Enter numbers: "))
print(A)
# 5;-2;4
B = np.mat(input("Enter value matrix: "))
print(B)
```

```
Enter numbers: 1,-1,-2;1,-2,1;-2,1,1
[[ 1 -1 -2]
 [ 1 -2  1]
 [-2  1  1]]
Enter value matrix: 5;-2;4
[[ 5]
 [-2]
 [ 4]]
```

3

In [1]:

```
import numpy as np
X = np.mat(input("Enter numbers: "))
print(X)
X_trans = np.transpose(X)
print(X_trans)
for i in range(0,len(X)):
    for j in range(0,len(X)):
        if(X[i][j]==-X_trans[i][j]):
            print("The matrix is skew symmetric")
            break
        else:
            print("The matrix is not skew symmetric")
#0,-6,4;-6,0,7;4,7,0
```

```
Enter numbers: 0,-6,4;-6,0,7;4,7,0
[[ 0 -6  4]
 [-6  0  7]
 [ 4  7  0]]
[[ 0 -6  4]
 [-6  0  7]
 [ 4  7  0]]

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-1-2341ff038e9c> in <module>()
      4 X_trans = np.transpose(X)
      5 print(X_trans)
----> 6 for i in range(0,X.shape):
      7     for j in range(0,X.shape):
      8         if(X[i][j]==-X_trans[i][j]):

TypeError: 'tuple' object cannot be interpreted as an integer
```

4

In [14]:

```
print("------------------------------------")
A = np.mat(input("Enter the matrix: "))
print(A)
ev, evec = np.linalg.eig(A)
print("------------------------------------")
print("The eigen values of the matrix are:")
print(ev.round())
print("------------------------------------")
print("The eigen vectors of the matrix are: ")
print(evec.round())
#1,2,3;4,5,6;7,8,9
```

```
------------------------------------
Enter the matrix: 1,2,3;4,5,6;7,8,9
[[1 2 3]
 [4 5 6]
 [7 8 9]]
------------------------------------
The eigen values of the matrix are:
[ 16.  -1.  -0.]
------------------------------------
The eigen vectors of the matrix are:
[[-0. -1.  0.]
 [-1. -0. -1.]
 [-1.  1.  0.]]
```

Conclusion ~ The above results are the eigen vectors and values of the given matrix

5

In [25]:

```
# 1,2,9,12;4,4,8,14;8,6,7,15;9,7,13,18
Y = np.mat(input("Enter the matrix: "))
print(Y)
print(Y[0::2,0:2])
print(Y[1:,:2])
print(Y[0::2,1::2]) # 0 to end, skipping 1
print(Y[1:3,2:5])
print(Y[:,2:])
```

```
Enter the matrix: 1,2,9,12;4,4,8,14;8,6,7,15;9,7,13,18
[[ 1  2  9 12]
 [ 4  4  8 14]
 [ 8  6  7 15]
 [ 9  7 13 18]]
[[1 2]
 [8 6]]
[[4 4]
 [8 6]
 [9 7]]
[[ 2 12]
 [ 6 15]]
[[ 8 14]
 [ 7 15]]
[[ 9 12]
 [ 8 14]
 [ 7 15]
 [13 18]]
```

Conclusion ~ These are the matrices which needed to be sliced from the original matrix

LAB 4

Linear Combination of vectors

In [1]:

```
# from numpy import *
# import numpy as np
# import numpy
```

Aim ~ To check if vector can be expressed in the form of 3 other vectors

In [1]:

```
import numpy as np
from sympy import *
a = np.zeros((3,3))
for i in range(1,4):
    print("Enter the elements of Vector ", i)
    for j in range(1,4):
        a[j-1][i-1]=int(input("Enter value"))

A=np.matrix(a)
b = np.zeros((3,1))
print("Enter the solution Vector")
for i in range(1):
    for j in range(1,4):
        b[j-1][i-1]=int(input("Enter value"))

soln = np.linalg.solve(A,b)
print ("\na1 = ",soln[0])
print ("\na2 = ",soln[1])
print ("\na3 = ",soln[2])
```

```
Enter the elements of Vector  1
Enter value1
Enter value2
Enter value3
Enter the elements of Vector  2
Enter value1
Enter value1
Enter value3
Enter the elements of Vector  3
Enter value1
Enter value4
Enter value8
Enter the solution Vector
Enter value14
Enter value20
Enter value40

a1 =  [ 7.2]

a2 =  [ 7.2]

a3 =  [-0.4]
```

In [2]:

```
a = np.zeros((3,3))
for i in range(1,4):
    print("Enter the elements of Vector ", i)
    for j in range(1,4):
        a[j-1][i-1]=int(input("Enter value"))

A=np.matrix(a)
b = np.zeros((3,1))
print("Enter the solution Vector")

for i in range(1):
    for j in range(1,4):
        b[j-1][i-1]=int(input("Enter value"))
        c = np.zeros((3,1))
        print("Enter the scalars")

for i in range(1):
    for j in range(1,4):
        c[j-1][i-1]=int(input("Enter value"))
        soln = np.linalg.solve(A,b)

if(np.allclose(soln,c)):
    print("Solution is satisfied")
else:
    print("Solution is not satisfied")
```

```
Enter the elements of Vector  1
Enter value1
Enter value2
Enter value4
Enter the elements of Vector  2
Enter value1
Enter value1
Enter value3
Enter the elements of Vector  3
Enter value1
Enter value4
Enter value8
Enter the solution Vector
Enter value14
Enter the scalars
Enter value20
Enter the scalars
Enter value40
Enter the scalars
Enter value14
Enter value14
Enter value14
Solution is not satisfied
```

Conclusion ~ The solution is not satisfied

Linear Transformations

1. Verify if a given transformation is linear or not with given vectors

2. To visualise the transformation

3. Find rank and nullity of a transformation (Kernel, range and nullity)

1.

L.H.S ~

$T(v_1 + v_2) = T((1, 2) + (2, 1))$

$= T((1 + 2), (2 + 1)) = T(3, 3) = (3, 3, 6)$

R.H.S ~

$T(v_1) + T(v_2) = T(1, 2) + T(2, 1)$

$= (1, 2, 4) + (2, 1, 2)$

$= (3, 3, 6)$

In [2]:

```
v1 = [1,2]
v2 = [2,1]
res = []

def T(x1,y1):
    return (v1[0],v1[1],(v1[0]*v1[1])*v1[1])

def T2(x2,y2):
    return (v2[0],v2[1],(v2[0]*v2[1]))

def T3(x1,y1):
    return(v1[0]+v1[1],v2[0]+v2[1])

l1 = T(1,2)
print("1st Transformation in R.H.S is: {0}".format(l1))

l2 = T2(2,1)
print("2nd Transformation in R.H.S is: {0}".format(l2))

for i in range(0,len(l1)):
    res.append(l1[i] + l2[i])

print("R.H.S Result is: {0}".format(tuple(res)))

l3 = T3(1,2)
print("1st Transformation in L.H.S is: {0}".format(l3))

def T4(l3):
    return(l3[0],l3[1],l3[0]+l3[1])

l4 = T4(l3)
print("L.H.S Result is: {0}".format(l4))

if(l4==tuple(res)):
    print("L.H.S = R.H.S")
else:
    print("L.H.S NOT EQUAL TO R.H.S")
```

```
1st Transformation in R.H.S is: (1, 2, 4)
2nd Transformation in R.H.S is: (2, 1, 2)
R.H.S Result is: (3, 3, 6)
1st Transformation in L.H.S is: (3, 3)
L.H.S Result is: (3, 3, 6)
L.H.S = R.H.S
```

$$T_1\ :\ R^3 -> R^3$$

$$T_1(x, y, z) = (x + y, y + z, z + x)$$

$$T_2(x, y, z) = (2x + y, 2y - 3x, x - z)$$

$$T_3(x, y, z) = (x + y, x - y, z)$$

$T_2 : R^2 -> R^3$

$T_4(x, y) = (x, y, 2y)$

$T_5(x, y) = (x - y, x + y, x)$

$T_6(x, y) = (x - y, 3y, 4x + 5y)$

$T_3 : R^2 -> R^2$

$T_7(x, y) = (-y, x)$

$T_8(x, y) = (x + y, y)$

In [ ]:

```
from sympy import *
from numpy import *
dimension = int(input("Enter the dimension of the 1st domain"))
x_array=[]
y_array=[]
l=[]
for i in range(2):
    x = int(input("Enter the x value"))
    y = int(input("Enter the y value"))
    l.append(trans(x,y))
    x_array.append(x)
    y_array.append(y)

print("x_array",x_array)
print("y_array",y_array)
print("l array",l)
if sum(l)==sum(trans(sum(x_array),sum(y_array))):
    print(sum(l))
    print(sum(trans(sum(x_array),sum(y_array))))
    print("You are correct")

def trans(x,y):
    a = x
    b = y
    c = x+2*y
    return a,b,c
```

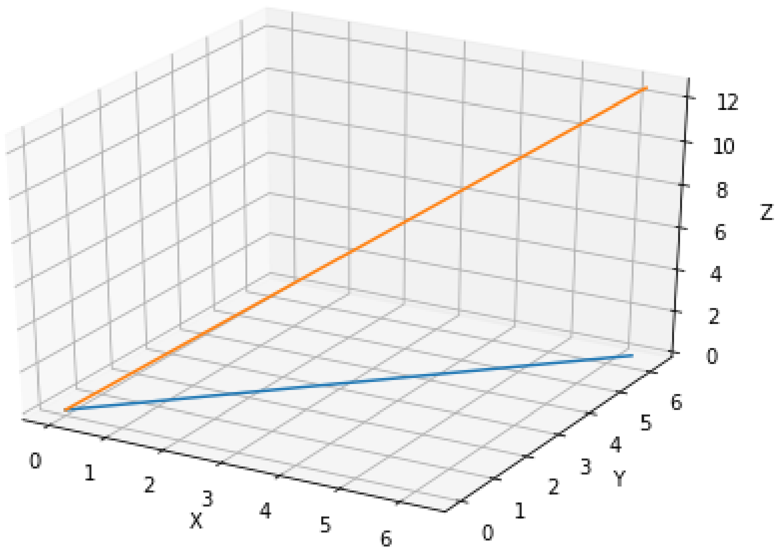
2.

In [24]:

```
import matplotlib.pyplot as plt
from pylab import *
from mpl_toolkits.mplot3d import Axes3D
ax = Axes3D(figure())

def f(x,y,z):
    ax.plot(x,y,z)
    z = x + y
    ax.plot(x,y,z)
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
    show()

x = linspace(0,2*pi,400)
y = linspace(0,2*pi,400)
z=0
f(x,y,z)
```



$$T(x + y) = T(x) + T(y)$$

$$T(ax) = aT(x)$$

Therefore, the transformation is plotted in 2 -d.