MODULE:2
NUMPY MODULE-ARRAYS AND MATRICES

The *numpy* module supports operations on compound data type like *arrays()* and *matrices()*. i.e., arrays and matrices can be created using numpy module. Python lists can be converted to multidimensional arrays also. There are several other functions that can be used for creating matrices . The mathematical functions like sine,cosine,etc., of numpy accepts array objects as arguments and return the results as arrays objects. Numpy arrays can be indexed, sliced and copied like python lists.

Eg:1 To make an array from python lists

```
from numpy import*
x=[1,2,3,4,5]
y=array(x)
print y,type(y)
```

or

```
from numpy import*
y=array([1,2,3,4,5])
print y,type(y)
```

verify the o/p

```
#Eg: 2. numpy supports array objects as arguments for sine,cosine,etc.,
from numpy import*
x=array([0,pi/4,pi/2, 3*pi/4,2*pi])
print sin(x),cos(x)
```

CREATING MULTI-DIMENSIONAL ARRAYS AND MATRICES

We can also make multi-dimensional arrays . Remember that a member of a list can be another list. The following example shows how to make a two dimensional array.

```
#Eg;3  creation of two dimensional arrays.
From numy import*
a=[[1,2],[3,4]]  #make a list of lists
x=array(a) #to convert into an array
print x
```

other than arrays(), there are several other functions that can be used for creating different types of arrays() and matrices(). Some of them are described below:

(a) *arange(start,stop,step,dtype=none)*

This function creates an evenly spaced one-dimensional array. Start,stop,step,dtype(data type) are the arguments. If dtype is not given it is deduced from other arguments. Note that, the values are generated within the interval, including start but excluding stop.

```
from numpy import*
arange(2.0,3.0,.1)  makes the array([2., 2.1,2.2,.....2.8,2.9])
special case: arange(20) means  array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
      17, 18, 19])
```

(*b) linspace(start,stop,number of elements)*

Similar to arange(), start,stop,no.of samples are the arguments.

```
from numpy import*
linspace(1,2,11)  is equivalent to array([1.,1.1,1.2,1.3,1.4,.....2.])
```

(c) zeros(shape,datatype)

This function returns a new array of given shape and type, filled with zeros. The arguments are shape and datatype. For eg., zeros([3,2],'float') generates a 3x2 array(3 rows x 2 columns) filled with zeros as shown below. If the datatype is not specified, it defaults to float.

```
array([[ 0.,  0.],
      [ 0.,  0.],
      [ 0.,  0.]])
special case:zeros(5) means     array([ 0.,  0.,  0.,  0.,  0.])
```

(d) ones(shape,datatype)

similar to zeros() ,except that the values are initiated by 1's

```
eg:ones([3,2])

array([[ 1.,  1.],
      [ 1.,  1.],
      [ 1.,  1.]])
special case: ones(60) means  array([ 1.,  1.,  1.,  1.,  1.,  1.])
```

(e) random.random(shape)

similar to the functions above, but matrix is filled with random numbers ranging from to 0 *to 1 of float*

type. For eg*:,random.random([3,3])*will generate a 3x3 matrix filled with random numbers .

```
array([[ 0.36718021,  0.35983883,  0.53850979],
       [ 0.90204303,  0.32407118,  0.90023759],
       [ 0.60699267,  0.36765672,  0.2608889 ]])
```

(f) *reshape(array,newshape)*

We can also make multi-dimensional arrays by reshaping a one-dimensional array. The function *reshape()* changes dimensions of an array. The total number of elements must be preserved.

For eg: from numpy import*

```
a=arange(20)
b=reshape(a,[4,5])
print b
```

the o/p will be

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]
```

(*g) copying*

Numpy arrays can be copied using copy method.

For eg:

```
from numpy import*
a=zeros(5)
print a ,'a array'
b=a.copy()
print a,b,' a array and b array'
a[0]=10
print a,b,'a array and b array after modification of a array'
```

the out put will be

```
 0.  0.  0.  0.  0.] a array
[ 0.  0.  0.  0.  0.] [ 0.  0.  0.  0.  0.]  a array and b array
[ 10.   0.   0.   0.   0.] [ 0.  0.  0.  0.  0.] a array and b array after modification of b array
```

note that ,eventhough b array is a copy of a array, any modification in the a array will not affect b array. If you want to make modifiction in b array, while modifying a array, use b=a

eg:

or eg:

```
from numpy import*
a=zeros(5)
print a ,'a array'
b=a
print a,b,' a array and b array'
a[0]=10
print a,b,'a array and b array after modification of a array'
```

the out put will be

[0. 0. 0. 0. 0.] a array

[0. 0. 0. 0. 0.] [0. 0. 0. 0. 0.] a array and b array

[10.   0.   0.   0.   0.] [10.   0.   0.   0.   0.] a array and b array after modification of b array

(h) creating identity matrix:

An identity matrix can be created by if the function identity(a,dtype),here 'a' is the dimension of the matrix. If the dtype is not specified, it is assumed to be float

for eg; I=identity(3) will create 3x3 identity elements(float type) matrix.

The o/p will be

```
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

(i) ones_like():

```
from numpy import*
a=array([[1,2],[3,4]])
b=ones_like(a)
```

will create a new matrix b having same dimensions as that of matrix but the elements are 1's

(j) changing elements of matrix:

```
from numpy import*
a=array([[1,2],[3,4]])
a[0]=100
```

print a

will replace all elements of the first row of matrix a with 100 .However, if we write a[0,0]=100 will replace the first element of the zeroth row and zeroth column having index a[0,0] with 100.

eg:

```
from numpy import*
a=array([[1,2],[3,4]])
print a
a[0]=100
print a
a=array([[1,2],[3,4]])
a[0,0]=100
print a
```

the o/p will be

```
[[1 2]
 [3 4]]
[[100 100]
 [  3   4]]
[[100   2]
 [  3   4]]
```

Transpose of a matrix

The transpose of a matrix is obtained by interchanging it elements.The transpose of a matrix A is given by A(i,j)=A(j,i) for all i's and j's

eg:

```
from numpy import*
a=array([[1,2],[3,4]])
print a
print a.T
```

or

```
transpose(a)
```

the o/p will be

```
[[1 2]
 [3 4]]
[[1 3]
```

[2 4]]

Arithmetic operations

Arithmetic operations performed on an array is carried out on all individual elements. Adding or multiplying array object with a number will multiply all the elements by that number. However, adding or multiplyig two arrays having identical shapes will result in performing that operation with the corresponding elements .

Eg:

```
from numpy import*
a=array([[2,3],[4,5]])
b=array([[1,2],[3,0]])
print a+b
print a*b
```

the out put will be

```
[[3 5]
 [7 5]]
[[ 2  6]
 [12  0]]
```

Matrix multiplication:

However, the matrix multiplication in the above eg.is not in the same way as we do in matrix algebra. To get the correct result, we convert the array in an actual matrix using *mat()* function

Eg;

```
from numpy import*
a=array([[2,3],[4,5]])
b=array([[1,2],[3,0]])
mata=mat(a)
matb=mat(b)
print mata+matb
print mata*matb
```

the o/p will be

```
[[3 5]
 [7 5]]
[[11  4]
 [19  8]]
```

cross product:

The *cross()* function returns the cross product of two vectors,defined by

AxB= i j k

A1 A2 A3 = i(A2B3-A3B2)-j(A1B3-A3B1)+k(A1B2-A2B1)

B1 B2 B3

this can be evaluated using the function cross(A,B).

for eg:

```
from numpy import*
a=array([1,2,3])
b=array([4,5,6])
c=cross(a,b )
print c
```

the o/p will be [-3 6 -3]

dot product:

The dot() returns the dot product of two vectors,defined by

A.B=A1B1+A2B2+A3B3

for eg:

```
from numpy import*
a=array([1,2,3])
b=array([4,5,6])
c=dot(a,b )
print c
```

the o/p will be 32

Saving and Restoring;

An array can be saved to text file using *array.tofile(file name)* and it can bbe read back using *array=fromfile()* methods, as show below:

```
from numpy import*
a=arange(10)
a.tofile('myfile.dat')
```

b=fromfile('myfile.dat',dtype='int')

print b

the output will be [0 1 2 3 4 5 6 7 8 9]

NOTE;

The function fromfile() sets dtype='float' by default. In this case we have saved an integer array and need to specify that while reading the file.

Matrix invertion:

The function *linalg.inv(name of matrix)* computes the inverse of a square matrix, if it exits(for non singular martix, det !=0)

if A= 0 1 2

1 2 3

3 1 1

the inverse of this matrix can be computed by the following python program

```
from numpy import*
A=array([[0,1,2],[1,2,3],[3,1,1])
Ainv=linalg.inv(A)
print Ainv
```

the output will be

```
[[ 0.5 -0.5  0.5]
 [-4.   3.  -1. ]
 [ 2.5 -1.5  0.5]]
```

we can verify this result by multiplying the original matrix A with its inverse (Ainv) giving a identity matrix. i.e, $AA^{-1}=I$

for eg:

```
from numpy import*
A=array([[0,1,2],[1,2,3],[3,1,1])
Ainv=linalg.inv(A)
print Ainv
print dot(A,Ainv)
```

Solution of simultaneous equations:

Consider the matrix eqn. AX=B , where A= $\begin{pmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \\ 3 & 1 & 1 \end{pmatrix}$ x= $\begin{pmatrix} x1 \\ x2 \\ x3 \end{pmatrix}$ B=$\begin{pmatrix} b1 \\ b2 \\ b2 \end{pmatrix}$

this eqn. Is equivalent to the system of three linear eqns. x2+2x3=b1,x1+2x2+3x3=b2,3x1+x2+x3=b3

solving for x1,x2 and x3 , we get, x1=1/2(b1-b2+b3),x2=-4b1+3b2-b3, x3=1/2(5b1-3b2+b3)

These are equivalent to a single matrix eqn.

$$\begin{pmatrix} x1 \\ x2 \\ x2 \end{pmatrix} = \begin{pmatrix} 1/2 & -1/2 & 1/2 \\ -4 & 3 & -1 \\ 5/2 & -3/2 & 1/2 \end{pmatrix} \begin{pmatrix} b1 \\ b2 \\ b3 \end{pmatrix}$$

i,e X= $A^{-1}B$

where A^{-1}= $\begin{pmatrix} 1/2 & -1/2 & 1/2 \\ -4 & 3 & -1 \\ 5/2 & -3/2 & 1/2 \end{pmatrix}$

eg:1

Write a Python program solve the system of equations, using matrix invertion method.

4x1-2x2+x3=11

-2x1+4x2-2x3=-16

x1-2x2+4x3=17

solution:

These equations are equivalent to the single matrix equation.

$$\begin{pmatrix} 4 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 4 \end{pmatrix} \begin{pmatrix} x1 \\ x2 \\ x3 \end{pmatrix} = \begin{pmatrix} 11 \\ -16 \\ 17 \end{pmatrix}$$

the above equation is of the form AX=B

therefore,

$X=A^{-1}B$

program

```
from numpy import*
A=array([[4,-2,1],[-2,4,-2],[1,-2,4]])
Ainv=linalg.inv(A)
B=array([11,-16,17])
print dot(Ainv,B)
```

eg:2

Write a Python program solve the system of equations, using matrix invertion method.

x1+x2+x3+x4=0

x1+x2+x3-x4=4

x1+x2-x3+x4=4

x1-x2+x3-x4=2