1. numpy

2. scipy

3. matplotlib (visualization)

matlab => technical computing software

Numpy:

1. array operations (multidimensional)

2. shaping

3. linear algebra

c:\python37\scripts>pip install numpy

Numpy Tutorial:

ndarray object => n dimensional array object

1. creating an array using numpy:

x=np.array([1,2,3,4,5,6])

y=np.array([1,2,3,4,5,6]).reshape(2,3)

z=np.array([1,2,3,4,5,6]).reshape(3,2)

2. using arange() method

a=np.arange(6)

b=np.arange(0,9,2)

3. creating multidimensional arrays directly and reverting back to 1d:

p=np.array([1,2,3,4,5,6]).reshape(2,3)

q=np.ravel()

4. Atrributes of ndarray object:

x.ndim => show the dimensions of the array

x.shape => show the shape of the array

x.size => no. of elements

x.T => Transpose (rows into columns and vice versa)

x.itemsize => gives the no. bytes occupied by each element

x.dtype=> data type of each element

x.dtype.name => displays the name of the data type

5. Creating unit matrix/zero values matrix

m=np.zeros((3,4), dtype=int)

n=np.ones((3,4), dtype=float)

6. Matrix Operations:

+

-

\*

/

sum()

sum(axis=0)

sum(axis=1)

(all these are direct operations)

r=x.dot(y)

matrix multiplication

7. Creating different dimensions of arrays:

import numpy as np

a=np.array([1,2,3])

b=np.array([[1,2,3],[4,5,6]])

c=np.array([[[1,2,3],[4,5,6]],[[6,7,8],[9,9,9]]])

d=np.array([1,2,3,4,5],ndmin=5)

8. Accessing array elements:

a[0]

a[1]

a[-1]

b[1,0]

b[0,2]

c[1,1,2]

9. slicing the arrays:

a[1:]

a[:3]

a[-3:-1]

a[1:6:2] step

b[1,1:3]

b[1:3,1]

b[1:3,1:3]

10. Type casting:

from float to int:

x=np.array([1,2,3,4,5,6],dtype=float)

x.dtype

y=x.astype('int') converting to int

z=y.astype(float)

11. copy and view:

copy is new array, view is part or full of the original array

a=np.arange(1,6)

b=a.copy()

c=a.view()

c[2]

c[2]=100

a[2]

12. check if the array owns the data

a=np.arange(6)

b=a.copy()

c=a.view()

print (b.base)

print (c.base)

13. Iterating thru the arrays

a=np.arange(6)

for x in a:

print (x)

arr = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])

for x in np.nditer(arr):

print(x)

for idx, x in np.ndenumerate(arr): with index number

print(idx, x)

14. Joining numpy arrays:

a=np.arange(1,6)

b=np.arange(7,12)

c=np.concatenate((a,b))

arr1 = np.array([[1, 2], [3, 4]])

arr2 = np.array([[5, 6], [7, 8]])

arr = np.concatenate((arr1, arr2), axis=1)

x=np.stack((arr1,arr2),axis=1)

y=np.vstack(arr1,arr2)

z=np.hstack(arr1,arr2)

15. Splitting the arrays:

b=np.array\_split(a,2) splitting into 2 arrays

b=np.array\_split((a,2),axis=1)

16. Searching for arrays:

a=np.arange(6)

b=np.where(a==4)

c=np.where(a%2==0)

17. Sorting the array:

a=np.array([3,2,4,6,51])

b=np.sort(a)

18. filtering the array:

a=np.array([3,2,4,6,51])

filter\_arr = arr > 42

newarr = arr[filter\_arr]

19. Universal Functions (ufuncs):

ufuncs stands for "Universal Functions" and

they are NumPy functions that operates on the ndarray object

Why use ufuncs?

ufuncs are used to implement vectorization in NumPy which is

way faster than iterating over elements.

They also provide broadcasting and additional methods like

reduce, accumulate etc. that are very helpful for computation.

ufuncs also take additional arguments, like:

'where' boolean array or condition defining where the operations should take place.

'dtype' defining the return type of elements.

'out' output array where the return value should be copied

What is Vectorization?

Converting iterative statements into a vector based

operation is called vectorization.

It is faster as modern CPUs are optimized for such operations.

ADDING ELEMENTS OF TWO LISTS:

x = [1, 2, 3, 4]

y = [4, 5, 6, 7]

z = []

for i, j in zip(x, y):

z.append(i + j)

print(z)

import numpy as np

x = [1, 2, 3, 4]

y = [4, 5, 6, 7]

z = np.add(x, y)

print(z)

'''

z=np.subtract(x,y)

z=np.multiply(x,y)

z=np.divide(x,y)

z=np.power(x,y)

z=np.mod(x,y)

z=np.divmod(x,y) => returns 2 arrays one for quotient, one for mod

z=np.absolute(x)

z=np.trunc(x)

z=np.fix(x)

z=np.round(x,2)

z=np.floor(x)

z=np.ceil(x)

z=np.log2(x)

z=np.log10(x)

z=np.log(x)

z=cumsum(x)

z=cumprod(x)

z=np.lcm(x,y) x and y are variables

z=np.gcd(x,y,z) x,y,z are variables

import numpy as np

arr = np.array([3, 6, 9])

x = np.lcm.reduce(arr) to get the lcm of array

z= np.gcd.reduce(arr) to get the gcd of the array

HOW TO CREATE YOUR OWN FUNCTIONS:

To create you own ufunc, you have to define a function,

like you do with normal functions in Python, then you

add it to your NumPy ufunc library with the frompyfunc() method.

The frompyfunc() method takes the following arguments:

function - the name of the function.

inputs - the number of input arguments (arrays).

outputs - the number of output arrays

import numpy as np

def myadd(x, y):

return x+y

myadd = np.frompyfunc(myadd, 2, 1)

print(myadd([1, 2, 3, 4], [5, 6, 7, 8]))

20. SET OPERATIONS:

What is a Set

A set in mathematics is a collection of unique elements.

Sets are used for operations involving frequent intersection, union and difference operations.

Create Sets in NumPy

We can use NumPy's unique() method to find unique elements from any array.

E.g. create a set array, but remember that the set arrays should only be 1-D arrays.

import numpy as np

arr = np.array([1, 1, 1, 2, 3, 4, 5, 5, 6, 7])

x = np.unique(arr)

print(x)

import numpy as np

arr1 = np.array([1, 2, 3, 4])

arr2 = np.array([3, 4, 5, 6])

newarr = np.union1d(arr1, arr2)

print(newarr)

import numpy as np

arr1 = np.array([1, 2, 3, 4])

arr2 = np.array([3, 4, 5, 6])

newarr = np.intersect1d(arr1, arr2, assume\_unique=True)

print(newarr)

import numpy as np

set1 = np.array([1, 2, 3, 4])

set2 = np.array([3, 4, 5, 6])

newarr = np.setdiff1d(set1, set2, assume\_unique=True)

print(newarr)

import numpy as np

set1 = np.array([1, 2, 3, 4])

set2 = np.array([3, 4, 5, 6])

newarr = np.setxor1d(set1, set2, assume\_unique=True)

print(newarr)

TRIGONOMETRIC FUNCTIONS:

sin(), cos(), tan(), deg2rad(), rad2deg(), arcsin(), arccos(), arctan(), hypot(),

sinh(), cosh(), tanh(),