Numerical Analysis & Plotting

Numpy – Overview, Setup, Datatypes, Basic Operators, Indexing, Broadcasting, Matrix Operators.

- NumPy is a Python library.
- NumPy is used for working with arrays.
- NumPy is short for "Numerical Python".
- It also has functions for working in domain of linear algebra, fourier transform, and matrices.
- NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.

- In Python we have lists that serve the purpose of arrays, but they are slow to process.
- NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.
- The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.
- Arrays are very frequently used in data science, where speed and resources are very important.
- NumPy is a Python library and is written partially in Python, but most of the parts that require fast computation are written in C or C++.

Introduction

- Before using the objects and routines available in Numpy we have to write any one of the following command:
  - Import numpy
  - import numpy as np
  - from numpy import arrange
  - from numpy import *

Checking NumPy Version

```
import numpy as np

print(np.__version__)
```

Numpy-Array Creation from functions

- There are multiple ways to create array of single/multi dimensions through Numpy. Those are
  - array()
  - empty()
  - zeros()
  - ones()
  - arrange()
  - linspace()

Numpy-Array Creation using array()

- Using an array() function in NumPy we can create any dimension array as follows:
  - To create single dimensional array:

```
from numpy import *
a = array([1,4,3], dtype=float)
print(a)
```

  - To create multi dimensional array:

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

Check Number of Dimensions

NumPy Arrays provides the ndim attribute that returns an integer that tells us how many dimensions the array have.

Check how many dimensions the arrays have:

```
import numpy as np

a = np.array(42)

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

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

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

print(a.ndim)

print(b.ndim)

print(c.ndim)

print(d.ndim)
```

Numpy-Array Creation using empty()

- It creates an uninitialized array of specified shape and data type.

```
from numpy import *
a = empty((3,2),dtype=int)
print(a)
```

numpy.empty(shape, dtype=float, order='C')

shape	Shape of the empty array, e.g., (2, 3) or 2.	Required
dtype	Desired output data-type for the array, e.g, numpy.int8. Default is numpy.float64.	optional
order	Whether to store multi-dimensional data in row-major (C-style) or column-major (Fortran-style) order in memory.	optional

Numpy-Array Creation using zeros()

- It creates an array of zeros of specified shape and data type.

```
from numpy import *
a = zeros((3,2),dtype=float)
print(a)
```

Numpy-Array Creation using ones()

- It creates an array of ones of specified shape and data type.

```
from numpy import *
a = ones((3,2),dtype=float)
print(a)
```

Numpy-Array Creation using arange()

- The arange() function in numpy is same as range() function in Python. The arrange() function is used in the following format:
  - arange(start, stop, stepsize, dtype)
- This creates an array with a group of elements from ‘start’ to one element prior to ‘stop’ in steps of ‘stepsize’ having dtype as specified.

```
from numpy import *
a = arange(2,13,2,dtype=float)
print (a)
```

```
import numpy as np

print("A\n", np.arange(4).reshape(2, 2), "\n")
print("A\n", np.arange(4, 10), "\n")
print("A\n", np.arange(4, 20, 3), "\n")
```

Output:

```
A
 [[0 1]
 [2 3]]

A
 [4 5 6 7 8 9]

A
 [ 4  7 10 13 16 19]
```

Numpy-Array Creation using linspace()

- This function is similar to arange() function. In this function, instead of step size, the number of evenly spaced values between the interval is specified. The usage of this function is as follows –
  - linspace(start, stop, num, dtype)

```
from numpy import *
a = linspace(10,20,5, dtype=complex)
print (a)
```

Numpy-Array- Indexing & Slicing

- Contents of ndarray object can be accessed and modified by indexing or slicing.
- Items in ndarray object follows zero-based index.

```
import numpy as np

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

print(arr[0])
```

To access elements from 2-D arrays we can use comma separated integers representing the dimension and the index of the element.

```
import numpy as np

arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print('2nd element on 1st row: ', arr[0, 1])
```

- Access 3-D Arrays
- To access elements from 3-D arrays we can use comma separated integers representing the dimensions and the index of the element.

```
import numpy as np

arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])

print(arr[0, 1, 2])
```

Numpy-Array- Indexing & Slicing

- A Python slice object is constructed by giving start, stop, and step parameters to the built-in slice function.
- This slice object is passed to the array to extract a part of array.

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

Slicing arrays

- Slicing in python means taking elements from one given index to another given index.
- We pass slice instead of index like this: [start:end].
- We can also define the step, like this: [start:end:step].
- If we don't pass start its considered 0
- If we don't pass end its considered length of array in that dimension
- If we don't pass step its considered 1

```
import numpy as np

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

print(arr[1:5])
```

- Slice elements from index 4 to the end of the array:
- import numpy as np

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

  print(arr[4:])
  ```
- Slice elements from the beginning to index 4 (not included):
- import numpy as np

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

  print(arr[:4])
  ```

Negative Indexing

- Use negative indexing to access an array from the end.
- import numpy as np

```
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print('Last element from 2nd dim: ', arr[1, -1])
```

Slicing 2-D Arrays

From the second element, slice elements from index 1 to index 4 (not included):

- import numpy as np

```
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

print(arr[1, 1:4])
```

- From both elements, return index 2:
- import numpy as np

  arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

  print(arr[0:2, 2])

- From both elements, slice index 1 to index 4 (not included), this will return a 2-D array:
- import numpy as np

  arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

  print(arr[0:2, 1:4])

Output

[[2 3 4]
 [7 8 9]]

Reshaping arrays

- Reshaping means changing the shape of an array.
- The shape of an array is the number of elements in each dimension.
- By reshaping we can add or remove dimensions or change number of elements in each dimension

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

newarr = arr.reshape(4, 3)

print(newarr)
```

Numpy-Array- Manipulation

- NumPy provides reshape function to resize an array without changing its data.

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

Output:

```
[[1 2 3]
 [4 5 6]]

[[1 2]
 [3 4]
 [5 6]]
```

Flattening the arrays

Flattening array means converting a multidimensional array into a 1D array.
We can use `reshape(-1)` to do this.

Convert the array into a 1D array:

```
import numpy as np

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

newarr = arr.reshape(-1)

print(newarr)
```

Iterating Arrays

```
import numpy as np

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

for x in arr:
  print(x)
```

```
import numpy as np

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

for x in arr:
  print(x)
```

```
[1 2 3]
[4 5 6]
```

Joining NumPy Arrays

- Joining means putting contents of two or more arrays in a single array.
- import numpy as np

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

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

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

print(arr)
```

Output: [1 2 3 4 5 6]

- **import numpy as np**

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

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

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

print(arr)
```

Splitting NumPy Arrays

- Splitting is reverse operation of Joining.
- Joining merges multiple arrays into one and Splitting breaks one array into multiple.
- import numpy as np

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

  newarr = np.array_split(arr, 3)

  print(newarr)
- Output:[array([1, 2]), array([3, 4]), array([5, 6])]

Sorting Arrays

- import numpy as np

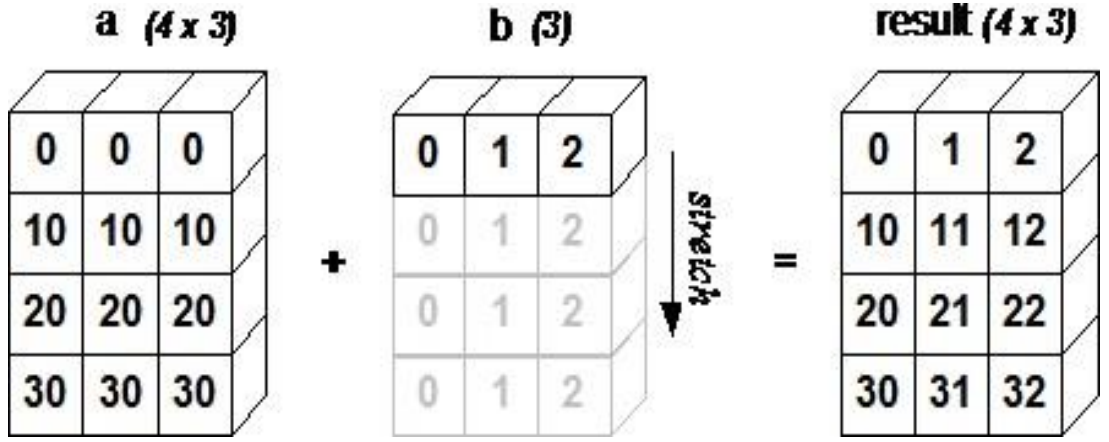
  arr = np.array([3, 2, 0, 1])

  print(np.sort(arr))

- Output: [0 1 2 3]
- **Note:** This method returns a copy of the array, leaving the original array unchanged.

Numpy-Array- Broadcasting

- The term **broadcasting** refers to the ability of NumPy to treat arrays of different shapes during arithmetic operations. Arithmetic operations on arrays are usually done on corresponding elements. If two arrays are of exactly the same shape, then these operations are smoothly performed. If the dimensions of two arrays are dissimilar, element-to-element operations are not possible. However, operations on arrays of non-similar shapes is still possible in NumPy, because of the broadcasting capability. The smaller array is **broadcast** to the size of the larger array so that they have compatible shapes. The following figure demonstrates how array **b** is broadcast to become compatible with **a.**



Numpy-Array- Broadcasting

```
import numpy as np
a = np.array([17, 11, 19]) # 1x3 Dimension array
print(a)
b = 3
print(b)
# Broadcasting happened because of miss match in array Dimension.
c = a + b
print(c)
output:
[17 11 19]
3
[20 14 22]
```

```
import numpy as np
A = np.array([[11, 22, 33], [10, 20, 30]])
print(A)
b = 4
print(b)
C = A + b
print(C)
```

Output:

```
[[11 22 33]
 [10 20 30]]
 4
[[15 26 37]
 [14 24 34]]
```

Numpy-Matrix Library

- NumPy package contains a Matrix library matlib. This module has functions that return matrices object instead of ndarray objects.
- Matrix is always two-dimensional, whereas ndarray is an n-dimensional array. This has the following functions
  - matlib.empty(): This function returns a new matrix without initializing the entries.
  - matlib.zeros(): This function returns the matrix filled with zeros.
  - matlib.ones(): This function returns the matrix filled with 1s.
  - matlib.eye(): This function returns a matrix with 1 along the diagonal elements and the zeros elsewhere.
  - matlib.identity(): This function returns the Identity matrix of the given size.
  - matlib.rand(): This function returns a matrix of the given size filled with random values.

Numpy-Matrix Library

```
import numpy.matlib
import numpy as np
print(np.matlib.empty((2,2)))
print(np.matlib.zeros((3,2)))
print(np.matlib.ones((3,2)))
print(np.matlib.eye(3,3))
print(np.matlib.identity(3))# Return a identity matrix i.e. a square matrix with ones on the main diagonal.
print(np.matlib.rand((3,3)))
```

Questions:

- Data Types in NumPy
- The Difference Between Copy and View
- Discuss numpy array search. How searchsorted() works.
- How do you sort a 2D array

```
import numpy as np
 # 2x2 matrix with 1's on main diagonal
b = np.identity(2, dtype = float)
print("Matrix b : \n", b)
a = np.identity(4)
print("\nMatrix a : \n", a)
```

Output :

```
Matrix b :
 [[ 1.  0.]
 [ 0.  1.]]
Matrix a :
 [[ 1.  0.  0.  0.]
 [ 0.  1.  0.  0.]
 [ 0.  0.  1.  0.]
 [ 0.  0.  0.  1.]]
```