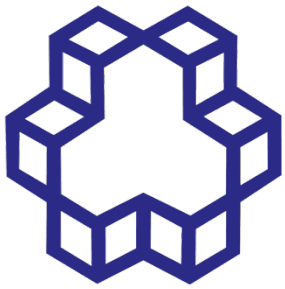

1928
K. N. Toosi University
of Technology

Department of electrical engineering

Course: Linear Algebra

Professor: Dr. Mehsan Tavakoli

Lab Instructions - session 1

Introduction to numpy and matplotlib

Deadline: -

Prepared by
Ramin Tavakoli

A review of numpy arrays and matrices + matplotlib

In numpy, the multidimensional array (N-dimensional array) or ndarray for short is the main tool for storing data and performing mathematical operations on it. A multi-dimensional array is an array that can have any number of dimensions, from a one-dimensional array (so-called vector) to multi-dimensional arrays such as a two-dimensional matrix and a three-dimensional cube or even more dimensions. In most data analysis and scientific calculations, these arrays are usually used in order to store and perform calculations optimally. One of the most important points about nampy arrays is that, unlike Python lists, the data stored in all the houses of a nampy array must be of the same type.

Open an interactive Python environment (python shell, ipython shell, Jupyter notebook or Colab notebook) or a Python IDE (IDLE editor, vs code, spyder, pyCharm, etc.)., run the following commands, and see the output.

1. Creating numpy arrays

```
import numpy
import numpy as np #Generally, Numpy library is conventionally used in
this way.
list = [1,2,3]
list
a = numpy.array(list)
a = np.array([1, 2, 3]) # 1_D array containing 3 numbers
a
a[1] = 39
a
print(type(list))
print(type(a))

a = np.array(list)
a
a = np.zeros(5)
a
a.dtype
a[1] = 4
a
a = np.zeros((5,), dtype=int)
a
a.dtype
a = np.ones(9)
print(a)
a = np.ones(12) * -20
print(a)
```

```
print(np.full(10, 222))
a = np.arange(10)
print(a)
print(2**a)
np.arange(0, 15, 4)
np.linspace(0, 20, 5)
np.eye(3)
np.empty(3)
```

2. Numpy array basic properties

```
import sys
a = np.ones(100)
print(len(a))
print(a.shape)
print(type(a))
print(a.size)
print(a.ndim)
print(a.dtype)
print(a.nbytes)
print(a.itemsize)
print(sys.getsizeof(a))
```

- Why are the outputs of a.nbytes and sys.getsizeof(a) different? (Note: Think before searching! ☺)
- What is the outputs of a.ndim?

Awnser:

3. Lists vs. numpy arrays

```
list1 = [0,2,4,5]
list2 = [9,5,2,4]
array1 = np.array(list1)
array2 = np.array(list2)
print(list1+list2)
print(array1+array2)
```

- Why are the outputs of the list1 + list2 and array1 + array2 different?
  You probably don't have an answer for that.

4. Data types

```
a = np.array([1,2,3,4,5,6,7,8,9,10], dtype=np.int64)
b = np.array(range(1, 11), dtype=np.int32)
c = np.array(range(1, 11), dtype=np.int16)
d = np.array(range(1, 11), dtype=np.int8)
e = np.array(range(1, 11), dtype=np.uint8)
print(a.itemsize, b.itemsize, c.itemsize, d.itemsize, e.itemsize)
print(a.nbytes, b.nbytes, c.nbytes, d.nbytes, e.nbytes)
d-4
e-4
f = np.array(range(1, 11), dtype=np.float32)
g = np.array(range(1, 11), dtype=np.float64)
h = np.array(range(1, 11), dtype=np.float64) # You may see an error.
print(f.nbytes, g.nbytes, h.nbytes)
l = np.array([False, True, True])
l.dtype
l = np.array([0, 1, 1], dtype=np.bool) # You may see an error.
l.dtype
l.nbytes
b = np.array([1, 2, 3], dtype=complex)
```

5. Basic operations with np-array

```
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
a+b
a-b
a*b       # we don't called this matrix multiplication
b**a      # we don't called this matrix multiplication
a + 4
a * 2
a ** 2
a.dtype
a/b      # different in pythons 2.x and 3.x
a//b
a = np.array([1.0, 2, 3])
a.dtype
a / b
a//b    # different in pythons 2.x and 3.x
a = np.array([1, 2, 3], dtype=np.float64)
a.dtype
```

6. 2D Arrays

```
a = np.array([[1, 2],
              [3, 4],
              [5, 6]], dtype='float64')
a
print('number of dimensions:', a.ndim)
print('shape:', a.shape)
print('size:', a.size)
print('dtype:', a.dtype)
np.array([[['000', '001'], ['010', '011']],
          [['100', '101'], ['110', '111']]])
A = np.zeros((4, 6))
A
A = np.zeros((4, 6), dtype=np.int32)
A
A = np.ones((3, 7))
A
A = np.ones((3, 8), dtype=np.uint8)
A
np.full((4, 3), 50.0)
a = np.random.randint(0, 10, (2, 3))
a = np.full_like(a, 3)
a
A = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
A[1, 2]
A[0, -1]
A[1, 2]
A.shape
A.shape[0]
A.shape[1]
A.shape[::-1]
A.size
A.ndim
A[0]
A[1]
A[0].shape
A[0, :]
A[0, :].shape
A[[0], :]
A[[0], :].shape
A[:, 2]
A[:, [2]]
A[:, 2].shape
```

```
A[:, [2]].shape
A[1:3]
A[1:3, :]
A[:, :3]
A[:, ::2]
A[:, ::-1]
r = np.array([0, 1, 0, 2, 2])
A
r
A[r, :]
A
A[:,0] = 1
A
A[:,0] = [20,30,40]
A
A.T
B = np.array([[1,1,1,1], [2,2,2,2], [3,3,3,3]])
A
B
A + B
A * B
A = A.reshape(4,3)
A.shape
np.dot(A,B)
A.dot(B)
A @ B
A.dot(B.T)  # Error ,use reshape
I = np.eye(3)
I
np.random.random((2,3))
np.random.random((2,3))
np.random.random((2,3))
np.random.rand(2,3)
np.random.rand(2,3)
np.random.randn(2,3)
np.random.randn(2,3)
np.random.randint(0, 10, (3, 3))
np.random.normal(0, 1, (2, 3))
```

- What is the differences between random functions?

7. Indexing and Slicing

```
a = np.array([0, 1, 2, 3, 4, 3, 2, 1, 0])
a
print('a[0] =', a[0])
print('a[2] =', a[2])
print('a[-1] =', a[-1])
print(a[:3])
print(a[2:4])
print(a[1:5:2])
print(a[::-1]) # ?
print('---')
a[2:-1]
a[2:]
print(a[2:5:-1])
a[8:2:-1]
a[::-1]
a = np.random.randint(0, 10, (3, 5))
print(a)
print('a[2, 3] =', a[2, 3])
a[2, 3] = 10
print('a[2, 3] =', a[2, 3])
a = np.array([[0, 1, 2],
              [3, 4, 5],
              [6, 7, 8],
              [9, 10, 11]])
print(a[:2, 1:])
print(a[1:3, ::2])
print(a[:2, 1])
print(a[2, :])
print(a[:, 1])
print(a[2])
```

8. Reshaping

```
a = np.arange(16)
print(a.reshape(4, 4))
print(a)    # what's happening?

a = np.array([1, 2, 3])
print(a.reshape(3, 1))
print(a.reshape(1, 3))
a = np.arange(12)
```

```
print(a.reshape(2, -1, 3))
'''
If we enter the size of exactly one of the dimensions equal to -1 to
reshape, then
Numpy will recognize the size of that dimension based on the size of
the array and
the size of the other input dimensions.
'''
a = np.array([[0, 1, 2],
              [3, 4, 5],
              [6, 7, 8]])
print(a.reshape(-1))

A = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
print('----')
A.reshape((4, 3))
A.reshape((2, 6))
# A.reshape((2, 7))
A.reshape((1, 12))
A.reshape((12, 1))
A.reshape((12,))
b = A.ravel()
print(b)
print(b.shape)
b.reshape((2, 6))
print(b)
b.shape = (2, 6)
print(b)
f = A.flatten()
r = A.ravel()
print(f)
print(r)
f[0] = 4444
print(f)
print(A)
r[0] = 4444
print(r)
print(A)
```

- What is the difference between ravel and flatten? Which one do you think is faster?

Read more:
[Does numpy reshape create a copy? Is there a way to do a reshape on numpy arrays but inplace?]

9. Concatenate

```
x = np.arange(6).reshape(2, 3)
y = np.arange(6, 12).reshape(2, 3)
z = np.arange(12, 15).reshape(1, 3)
np.concatenate([x, y, z])
np.concatenate([x, y], axis=1)

X = np.array([[1,2],
              [3,4]])
Y = np.array([[10,20,30],
              [40,50,60]])
Z = np.array([[7,7],
              [8,8],
              [9,9]])

print(np.concatenate((X,Z)))
print(np.concatenate((X,Z), axis=0))
print(np.concatenate((X,Y), axis=1))
print(np.vstack((X,Z)))
print(np.r_[X,Z])
# print(np.hstack(X,Y) # error
print(np.hstack((X,Y)))
print(np.c_[X,Y])
print(np.tile(Y,(4,3)))
```

10. Operations on arrays

```
A = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
A.sum()
A.sum(axis=0)
A.sum(axis=1)
A.min()
A.min(axis=0)
A.max(axis=0,keepdims=True)
A.max(axis=1)
A.max(axis=1,keepdims=True)
A.mean(axis=1)
A.prod(axis=0)
a = np.random.randint(0, 10, 4)
b = np.random.randint(0, 10, 4)
print(np.add(a, b), a + b, np.subtract(a, b), a-b)
a1, a2 = np.argmax(b), np.argmin(b)
```

11. Mask

```
A = np.array([[1, 2, 3, 4],
              [5, 6, 7, 8],
              [9, 10, 11, 12]])
Mask = np.array([[True, False, True, False],
                 [True, True, False, False],
                 [False, False, False, True]])
Mask.dtype
A
A[Mask]
A[~Mask]
A[Mask] = 222
A
A[~Mask] *= 2
A
A = np.array([[1, 2, 3, 4],
              [5, 6, 7, 8],
              [9, 10, 11, 12]])
B = np.zeros_like(A)
B[Mask] = A[Mask]
print(B)
A > 2
Mask = A < 8
Mask
A[Mask]
A[A < 8]
A[A < 8] += 100
A
```

12. Numpy slices are references (not copies)

```
A = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
b = A[:, 1]
print(b)
print(A)
b[1] = 10000
print(b)
print(A)
b = A[:, 0].copy()
print(b)
b[1] = -20000
print(b)
print(A)
```

13. Numpy arrays vs numpy matrices

```
A = np.array([[1, 2, 3],
              [1, 1, 1],
              [-1, -2, -1]])
A
A*A                 # element-wise multiplication
A.dot(A)          # matrix multiplication
A @ A               # matrix multiplication (same as above)

M = np.matrix([[1, 2, 3],
               [1, 1, 1],
               [-1, -2, -1]])
type(M)
M
M*M                          # matrix multiplication
np.multiply(M, M)            # element-wise multiplication
M = np.mat(A)
print(M)
M = np.matrix(A)
print(M)
print(M.T)
print(M.I)
M.I * M
M * M.I
M.A
type(M)
type(M.A)
C = np.matrix("1 2; 3 4; 5 6")
C
M*C
```

14. N-dimensional arrays

```
A = np.zeros((2,4,3))
A
A.shape
A[:,:,0].shape
A[:,:,0] = [[1,2,3,4],[5,6,7,8]]
A[:,:,1] = [[2,2,2,2],[4,4,4,4]]
A[:,:,2] = [[10,20,30,40],[11,21,31,41]]
A
A[0,:,:]
```

```
A[0]
A[:,1,:]
A[:,1]
A[:,[1]]
A[:,1].shape
A[:,[1]].shape
A[:,:,0]
A[:,:,2]
A[:,2:,2]
A.ravel()
```

15. Broadcasting

"In the future, we'll delve into this crucial topic together through an video discussion."

```
A = np.array([0, 1, 2, 3, 4, 5, 6, 7])
A - 10
a = np.array([4])
A * a
A = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
b = np.array([1, 0, 2, -2])
A
b
A-b
c = np.array([1, 2, 3])
c - c
c - c.reshape((3, 1))
c.reshape((-1, 1))
c - c.reshape((-1, 1))
c - c[:, np.newaxis]
c - c[:, None]
# A-c     # ValueError: operands could not be broadcast together with
shapes (3,4) (3,)
A-c.reshape((3, 1))
A = np.arange(24).reshape((2, 3, 4))
A.shape
A - 2
A - np.array([1, 2])
A - np.array([1, 2, 3, 4])
A - np.array([1, 2, 3])
A - np.array([1, 2, 3]).reshape((3, 1))
A - np.array([1, 2])
A - np.array([1, 2]).reshape((2, 1, 1))
A - np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
```

```
A - np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
A - np.array([[1, 2, 3, 4], [5, 6, 7, 8]]).reshape((2, 1, 4))
a = np.array([1, 2, 3, 4, 5])
b = np.array([1, 2, 3, 4]).reshape(4, 1)
print(a)
print(b)
a * b
c - c[:, None]
A-c
A-c.reshape((3,1))
A = np.arange(24).reshape((2,3,4))
A.shape
A - 2
A - np.array([1,2])
A - np.array([1,2,3,4])
A - np.array([1,2,3])
A - np.array([1,2,3]).reshape((3,1))
A - np.array([1,2])
A - np.array([1,2]).reshape((2,1,1))
A - np.array([[1,2,3,4],[5,6,7,8], [9,10,11,12]])
A - np.array([[1,2,3,4],[5,6,7,8]])
A - np.array([[1,2,3,4],[5,6,7,8]]).reshape((2,1,4))
```

To get a better understanding of broadcasting, read the following (particularly the broadcasting rules) https://numpy.org/doc/stable/user/basics.broadcasting.html

16. Math functions

```
x = np.arange(0, 2 * np.pi, 0.1)
x
y = np.cos(x)
y
np.sin(x)
np.tan(x)
x = np.linspace(1, 8, 20)
x
x.shape
np.exp(x)
np.log(x)
np.log10(x)
np.log2(x)
np.floor(x)
np.ceil(x)
```

```
np.round(x)
np.sqrt(x)
np.arctan(x)
```

- Here you can find a list of numpy math functions:
  - https://numpy.org/doc/stable/reference/routines.math.html

17. Plotting with Matplotlib

```
from matplotlib import pyplot as plt
x = np.arange(0, 2 * np.pi, 0.1)
print(x)
y = np.cos(x)
print(y)
plt.plot(x, y)
plt.show()
plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))
plt.show()
```

18. Reading and displaying images

```
from matplotlib import pyplot as plt

I = plt.imread('write_your_picture_name.jpg')
print(I.shape)
print(I.dtype)
print(I)
plt.imshow(I)
plt.show()
plt.imshow(I,cmap='gray')
plt.show()
plt.imshow(I[100:200,50:250],cmap='gray')
plt.show()
```

Reading and storing audio files

Read the following code (audio.py in the supplementary files) and run it.

audio.py

```
import numpy as np
import scipy.io.wavfile
import matplotlib.pyplot as plt

sampling_rate, data = scipy.io.wavfile.read('voice1.wav')

print(f"sampling rate: {sampling_rate}hz") # frequency (sample per
second)
print('data type:', data.dtype)
print('data shape:', data.shape)

# TODO: Calculate length of the signal in seconds
print(f"length = {length}s")

N, no_channels = data.shape # signal length and no. of channels

print('signal length:', N)

channel0 = data[:,0]
channel1 = data[:,1]

scale = 2**np.linspace(-2,4, N)

plt.plot(np.arange(N), scale)
plt.show()

print('shape_old:', scale.shape)
scale.shape = (N,1)
print('shape_new:', scale.shape)

data_new = np.int16(scale * data)

scipy.io.wavfile.write('output1.wav', sampling_rate, data_new)
```

- Play the audio file voice1.wav.
- Run the code to create output1.wav. Play the output audio file output1.wav.
- What is the above doing?
- What data type has been used for storing audio signals in a .wav file? Is it signed or unsigned?

- What are the variables N and sampling_rate. The audio signal has N samples played at sampling_rate samples per second. Calculate the length of the signal in seconds. Verify your answer by opening voice1.wav in an audio player.

- Why does the array data have two columns? What are the columns of data? Is this a mono or stereo audio?
- What is the array scale? How is it used here?
- Why did we change the shape of the scale array from N to (N,1)?
- What does the line data_new = np.int16(scale * data) do?
- Why did we cast the output data to int16 (16-bit signed integer)?

Task 1 - Practice Vectorization

Consider an arbitrary **A** matrix like

```
A = np.random.rand(200,10)
```

We perform the following operation on **A** to create the matrix **B**

```
mu = np.zeros(A.shape[1])
for i in range(A.shape[0]):
    mu += A[i]
mu /= A.shape[0]

B = np.zeros_like(A)
for i in range(A.shape[0]):
    B[i] = A[i] - mu
```

- What does the above piece of code do?
- Write an equivalent program **without loops**. Do it in just **a single line of code.**
- Write an equivalent program **with one loops**.

```
# TODO: equivalent program 1, Calculate B
# TODO: equivalent program 2, Calculate B
```

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ a_{31} & a_{32} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

Task 2

Plot the two channels of the input audio file (columns of the array data).

- Plot the two channels together in the same axes (like in the sin and cos example above)
- Analyse the audio signal. How does it correspond to what is said in the audio file?
- Use the zoom tool to zoom in the plot. How does an audio signal look like?
- Plot both channels using a single plt.plot function. This can be done by directly giving the array data as the second argument of plt.plot.
- Plot the channels of the output data (data_new). How has the shape changed compared to the input signal? Why?

Task 3

- Save the output audio using a different sampling rate. Try different choices such as sampling_rate*2 and sampling_rate//2. What happens?

Task 4

Change the file audio.py to, instead of scaling the signal, reverse it. Play the output to see how the audio sounds when played backwards.

- (Optional) Try using it on different audio files. Combine it with task 2. Have fun!

References

1. Numpy Quickstart https://numpy.org/doc/stable/user/quickstart.html
2. https://docs.scipy.org/doc/numpy-dev/user/numpy-for-matlab-users.html
3. https://docs.scipy.org/doc/numpy-dev/user/quickstart.html
4. Python Numpy Tutorial(with Jupyter and Colab)
5. Broadcasting