Part-II-B Basic NumPy Tutorials-II

March 1, 2025

---

1 Python : Numpy Tutorial

1.1 Introduction

Welcome to this NumPy tutorial!

NumPy (Numerical Python) is an essential Python library for numerical computations. This tutorial is designed specifically for medical researchers with no programming background. By the end of this tutorial, you will have a solid foundation in NumPy and be able to use it for your scientific research, including data analysis in the your domain.

1.1.1 Why Learn NumPy?

1. **Efficient Data Handling**: Perform fast mathematical operations on large datasets.
2. **Research Applications**: Useful for manipulating and analyzing research data, such as patient records, clinical trials, and imaging data.
3. **Integration**: Works seamlessly with other Python libraries like Pandas, Matplotlib, and SciPy.

1.1.2 Key Concepts Covered

In this tutorial, we will cover: 1. Basics of NumPy: Arrays, their creation, and manipulation. 2. Array operations: Mathematical, statistical, and logical. 3. Reshaping, indexing, and slicing arrays. 4. Working with real-world medical data examples. 5. Saving and loading data for reproducible research. 6. Interactive quiz to reinforce learning.

---

2 NumPy Functionality Table

Below is a comprehensive list of NumPy functions, their descriptions, and examples:

Function/Method	Description	Example	Output
`np.array()`	Creates an array from a list or tuple.	`np.array([1, 2, 3])`	`[1, 2, 3]`
`np.zeros()`	Creates an array of all zeros with a specified shape.	`np.zeros((2, 2))`	`[[0., 0.], [0., 0.]]`
`np.ones()`	Creates an array of all ones with a specified shape.	`np.ones((2, 3))`	`[[1., 1., 1.], [1., 1., 1.]]`
`np.arange()`	Creates an array with evenly spaced values within a range.	`np.arange(0, 10, 2)`	`[0, 2, 4, 6, 8]`
`np.linspace()`	Creates an array with evenly spaced values over a range.	`np.linspace(0, 1, 5)`	`[0., 0.25, 0.5, 0.75, 1.]`
`np.reshape()`	Reshapes an array to a new shape.	`np.array([1, 2, 3, 4]).reshape(2, 2)`	`[[1, 2], [3, 4]]`
`np.mean()`	Calculates the mean of array elements.	`np.mean([1, 2, 3, 4])`	`2.5`
`np.median()`	Calculates the median of array elements.	`np.median([1, 2, 3, 4])`	`2.5`
`np.std()`	Calculates the standard deviation.	`np.std([1, 2, 3, 4])`	`1.118`
`np.sum()`	Sums array elements.	`np.sum([1, 2, 3, 4])`	`10`
`np.max()`	Finds the maximum value in an array.	`np.max([1, 2, 3, 4])`	`4`
`np.min()`	Finds the minimum value in an array.	`np.min([1, 2, 3, 4])`	`1`
`np.sort()`	Sorts an array.	`np.sort([3, 1, 2])`	`[1, 2, 3]`
`np.unique()`	Finds unique elements in an array.	`np.unique([1, 2, 2, 3])`	`[1, 2, 3]`
`np.random.rand()`	Generates random numbers between 0 and 1.	`np.random.rand(2, 2)`	Random 2x2 array
`np.random.randint()`	Generates random integers within a range.	`np.random.randint(0, 10, (2, 2))`	Random integers 2x2 array
`np.linalg.solve()`	Solves a linear equation system.	`np.linalg.solve([[2, 3], [3, 2]], [8, 7])`	Solution array
`np.save()`	Saves an array to a binary file.	`np.save('file.npy', arr)`	Saves to file
`np.load()`	Loads an array from a binary file.	`np.load('file.npy')`	Loaded array

3 Topics to be Covered in This Tutorial

In the table above, we will explore a few commonly used functions as a starting point to avoid complexity. Later on, you can try them all on your own.

1. `np.array()`
2. `np.zeros()`
3. `np.ones()`
4. `np.arange()`
5. `np.linspace()`
6. `np.reshape()`
7. `np.mean()`
8. `np.median()`
9. `np.std()`
10. `np.sum()`
11. `np.max()`
12. `np.min()`
13. `np.sort()`
14. `np.random.rand()`
15. `np.linalg.solve()`
16. `np.save()`
17. `np.load()`

3.1 We will use simple examples to demonstrate how some of these function works...

---

4 Function Explanations and Examples

[ ]:
```

```

[12]:
```python
import numpy as np
```

4.0.1 1. Creating Arrays with `np.array()`

The `np.array()` function is used to create an array from a list or tuple. Arrays are the building blocks of NumPy.

[13]:
```python
# Example: Creating a 1D array
array = np.array([1, 2, 3, 4])
print("Array:", array)
```

```
Array: [1 2 3 4]
```

Explanation: - `np.array([1, 2, 3, 4])`: Converts the list `[1, 2, 3, 4]` into a NumPy array. - Output: `[1, 2, 3, 4]`

4.0.2 2. Creating an Array of Zeros with `np.zeros()`

This function creates an array filled with zeros of a specified shape.

[14]:
```
# Example: Creating a 2x3 array of zeros
zeros_array = np.zeros((2, 3))
print("Zeros Array:\n", zeros_array)
```

```
Zeros Array:
 [[0. 0. 0.]
 [0. 0. 0.]]
```

Explanation: - `np.zeros((2, 3))`: Creates a 2x3 array filled with zeros. - Output: `[[0., 0., 0.], [0., 0., 0.]]`

4.0.3 3. Creating an Array of Ones with `np.ones()`

This function creates an array filled with ones of a specified shape.

[15]:
```
# Example: Creating a 3x2 array of ones
ones_array = np.ones((3, 2))
print("Ones Array:\n", ones_array)
```

```
Ones Array:
 [[1. 1.]
 [1. 1.]
 [1. 1.]]
```

Explanation: - `np.ones((3, 2))`: Creates a 3x2 array filled with ones. - Output: `[[1., 1.], [1., 1.], [1., 1.]]`

4.0.4 4. Generating an Array with a Range of Values Using `np.arange()`

The `np.arange()` function creates an array with evenly spaced values within a given range.

[16]:
```
# Example: Generating an array from 0 to 9 with a step of 2
range_array = np.arange(0, 10, 2)
print("Range Array:", range_array)
```

```
Range Array: [0 2 4 6 8]
```

Explanation: - `np.arange(0, 10, 2)`: Generates values from 0 to 9 with a step of 2. - Output: `[0, 2, 4, 6, 8]`

4.0.5 5. Generating Evenly Spaced Values Using `np.linspace()`

The `np.linspace()` function generates evenly spaced numbers over a specified interval.

[17]:
```
# Example: Generating 5 evenly spaced values between 0 and 1
linspace_array = np.linspace(0, 1, 5)
print("Linspace Array:", linspace_array)
```

```
Linspace Array: [0.   0.25 0.5  0.75 1.  ]
```

Explanation: - `np.linspace(0, 1, 5)`: Generates 5 evenly spaced values between 0 and 1. - Output: `[0., 0.25, 0.5, 0.75, 1.]`

4.0.6 6. Reshaping Arrays with `np.reshape()`

The `np.reshape()` function reshapes an array to a specified shape.

[18]:
```
# Example: Reshaping a 1D array to 2x2
original_array = np.array([1, 2, 3, 4])
reshaped_array = original_array.reshape(2, 2)
print("Original Array:", original_array)
print("Reshaped Array:\n", reshaped_array)
```

```
Original Array: [1 2 3 4]
Reshaped Array:
 [[1 2]
 [3 4]]
```

Explanation: - `original_array.reshape(2, 2)`: Converts the 1D array `[1, 2, 3, 4]` to a 2x2 array. - Output: `[[1, 2], [3, 4]]`

4.0.7 7. Calculating the Mean with `np.mean()`

The `np.mean()` function calculates the mean of array elements

[19]:
```
# Calculating the mean of an array
data = np.array([1, 2, 3, 4, 5])
mean_value = np.mean(data)
print('Mean Value:', mean_value)
```

```
Mean Value: 3.0
```

Explanation: - `np.mean([1, 2, 3, 4, 5])`: Calculates the mean of the array elements. - Output: `3.0`

4.0.8 8. Calculating the Median with `np.median()`

The `np.median()` function calculates the median of the array elements. The median is the middle value when the data is sorted.

[20]:
```python
# Example: Calculating the median of an array
array = np.array([1, 2, 3, 4, 5])
median_value = np.median(array)
print("Median:", median_value)
```

```
Median: 3.0
```

Explanation: - `np.median([1, 2, 3, 4, 5])`: Calculates the median of the array elements. - Output: `3.0`

4.0.9 9. Calculating the Standard Deviation with `np.std()`

The `np.std()` function calculates the standard deviation of the array elements, which measures the spread or dispersion of the data.

[21]:
```python
# Example: Calculating the standard deviation of an array
array = np.array([1, 2, 3, 4, 5])
std_dev = np.std(array)
print("Standard Deviation:", std_dev)
```

```
Standard Deviation: 1.4142135623730951
```

Explanation: - `np.std([1, 2, 3, 4, 5])`: Calculates the standard deviation of the array elements. - Output: `1.4142135623730951`

4.0.10 10. Summing the Elements with `np.sum()`

The `np.sum()` function calculates the sum of all elements in the array.

[22]:
```python
# Example: Summing the elements of an array
array = np.array([1, 2, 3, 4, 5])
sum_value = np.sum(array)
print("Sum:", sum_value)
```

```
Sum: 15
```

Explanation: - `np.sum([1, 2, 3, 4, 5])`: Sums the array elements. - Output: `15`

4.0.11 11. Finding the Maximum Value with `np.max()`

The `np.max()` function returns the maximum value from the array.

[23]:
```python
# Example: Finding the maximum value in an array
array = np.array([1, 2, 3, 4, 5])
max_value = np.max(array)
print("Max:", max_value)
```

```
Max: 5
```

Explanation: - `np.max([1, 2, 3, 4, 5])`: Finds the maximum value in the array. - Output: `5`

4.0.12 12. Finding the Minimum Value with `np.min()`

The `np.min()` function returns the minimum value from the array.

[24]:
```python
# Example: Finding the minimum value in an array
array = np.array([1, 2, 3, 4, 5])
min_value = np.min(array)
print("Min:", min_value)
```

```
Min: 1
```

Explanation: - `np.min([1, 2, 3, 4, 5])`: Finds the minimum value in the array. - Output: `1`

4.0.13 13. Sorting the Array with `np.sort()`

The `np.sort()` function sorts the elements of the array in ascending order.

[25]:
```python
# Example: Sorting an array
array = np.array([5, 3, 1, 4, 2])
sorted_array = np.sort(array)
print("Sorted Array:", sorted_array)
```

```
Sorted Array: [1 2 3 4 5]
```

Explanation: - `np.sort([5, 3, 1, 4, 2])`: Sorts the array in ascending order. - Output: `[1, 2, 3, 4, 5]`

4.0.14 14. Generating Random Numbers with `np.random.rand()`

The `np.random.rand()` function generates random numbers from a uniform distribution between 0 and 1.

[26]:
```python
# Example: Generating a 2x2 array of random numbers
random_array = np.random.rand(2, 2)
print("Random Array:\n", random_array)
```

```
Random Array:
 [[0.64611576 0.14854119]
 [0.03401987 0.05335882]]
```

Explanation: - `np.random.rand(2, 2)`: Generates a 2x2 array with random values between 0 and 1. - Output: A 2x2 array with random values.

4.0.15 15. Solving Linear Systems with `np.linalg.solve()`

The `np.linalg.solve()` function solves a system of linear equations of the form Ax = B.

[27]:
```python
# Example: Solving a system of equations Ax = B
A = np.array([[3, 1], [1, 2]])
B = np.array([9, 8])
solution = np.linalg.solve(A, B)
print("Solution:", solution)
```

```
Solution: [2. 3.]
```

Explanation: - `np.linalg.solve(A, B)`: Solves the system of linear equations `A * x = B`. - Output: The solution for x.

4.0.16 16. Saving Arrays to a File with `np.save()`

The `np.save()` function saves an array to a binary file in `.npy` format.

[28]:
```python
# Example: Saving an array to a file
array = np.array([1, 2, 3, 4, 5])
np.save('array.npy', array)
print("Array saved to 'array.npy'")
```

```
Array saved to 'array.npy'
```

Explanation: - `np.save('array.npy', array)`: Saves the array to the file named 'array.npy'. - Output: A file is saved in the current directory.

4.0.17 17. Loading Arrays from a File with `np.load()`

The `np.load()` function loads an array from a file saved in `.npy` format.

[29]:
```
# Example: Loading an array from a file
loaded_array = np.load('array.npy')
print("Loaded Array:", loaded_array)
```

```
Loaded Array: [1 2 3 4 5]
```

Explanation: - `np.load('array.npy')`: Loads the array from the file 'array.npy'. - Output: The loaded array from the file. "'

4.1 Quiz

4.1.1 Test Your Knowledge

1. What function is used to create an array filled with zeros?
   - A) `np.zeros()`
   - B) `np.ones()`
   - C) `np.array()`
2. How do you reshape a 1D array into a 2D array?
   - A) `np.reshape()`
   - B) `np.array()`
   - C) `np.flatten()`

[ ]:
```

```

Main

March 1, 2025

1 Handling Images with NumPy

[25]:
```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
```

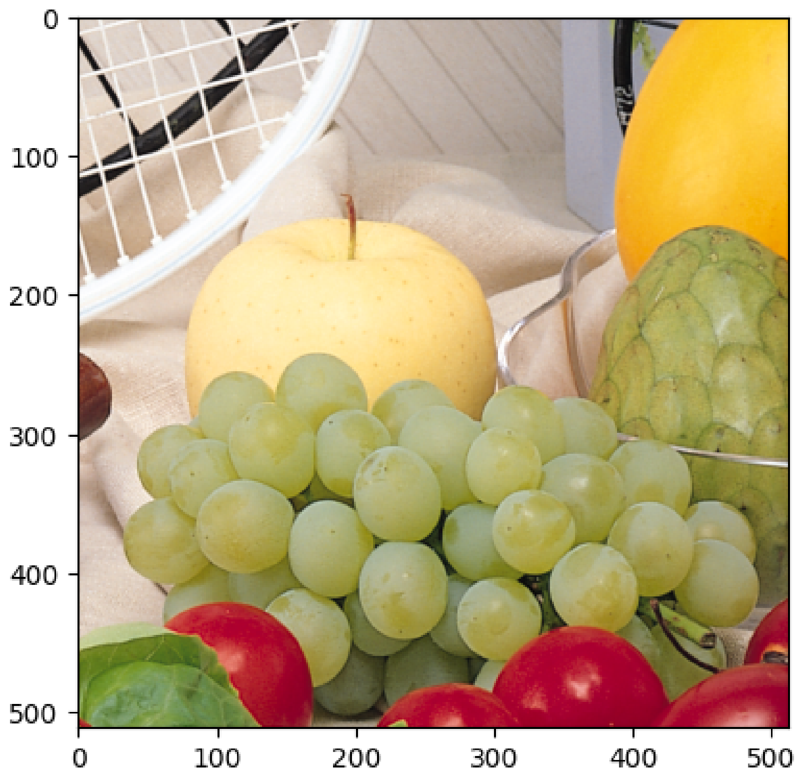
1. Open an image file
2. Read the image file into a numpy array

[26]:
```
png_img = cv.imread('standard_test_images/fruits.png', -1)
# png_img = cv.imread('x.png', -1)
## -1 flag is used to read the image as it is, without any changes
png_img.shape
```

[26]:
```
(512, 512, 4)
```

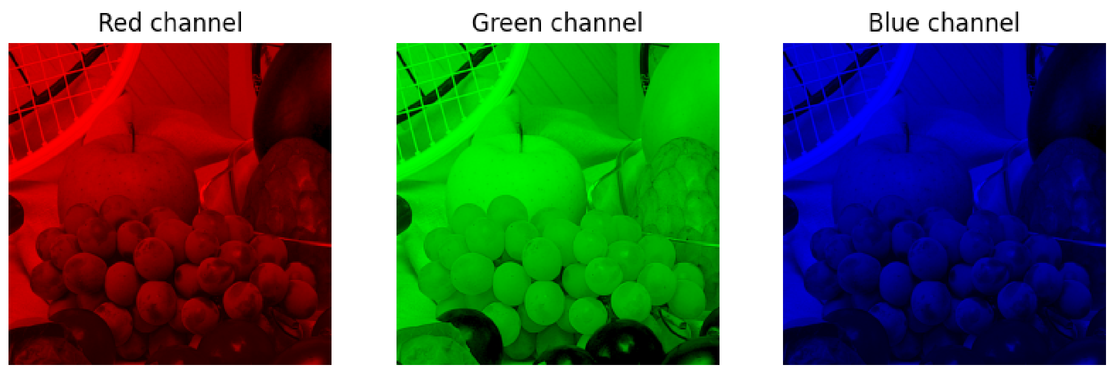
[27]:
```
# convert the image to RGB (OpenCV uses BGR)
png_img_rgb = cv.cvtColor(png_img, cv.COLOR_BGR2RGB)
print(png_img_rgb.shape)
# plot image inline with matplotlib
plt.imshow(png_img_rgb)
# plt.axis('off')
plt.show()
```

```
(512, 512, 3)
```



3. Filter np array to extract red | green | blue colors from the image and repaint image with only red component and display it

[28]:
```
# Ensure the image has 3 channels (RGB), ignoring alpha if present
if png_img_rgb.shape[-1] == 4:  # RGBA
    png_img_rgb = png_img_rgb[:, :, :3]

# create a placeholder image to store the image with only 0 values
zero_channel = np.zeros_like(png_img[:,:,0])

# split the image into its 3 channels
red_channel = png_img[:,:,0]
red_img = cv.merge((red_channel, zero_channel, zero_channel))

green_channel = png_img[:,:,1]
green_img = cv.merge((zero_channel, green_channel, zero_channel))

blue_channel = png_img[:,:,0]
blue_img = cv.merge((zero_channel, zero_channel, blue_channel))

# plot the 3 images in a subplot
plt.figure(figsize=(10,3))
plt.axis('off')

plt.subplot(131)
plt.imshow(red_img, cmap='gray')
plt.title('Red channel')
plt.axis('off')

plt.subplot(132)
plt.imshow(green_img, cmap='gray')
plt.title('Green channel')
plt.axis('off')

plt.subplot(133)
plt.imshow(blue_img, cmap='gray')
plt.title('Blue channel')
plt.axis('off')

# plt.axis('off')
plt.show()
```



Convert the image to grayscale and display it

[29]:
```
# using existing libaray function and calculate time taken
import time
start = time.time()
gray_img = cv.cvtColor(png_img, cv.COLOR_BGR2GRAY)
print(gray_img.shape)
end = time.time()
diff_lib = end - start
print(f'Time taken by OpenCV function: {diff_lib} seconds')
# the formula to compute average used here is 0.299*R + 0.587*G + 0.114*B
# because the human eye is more sensitive to green color, so it is given more␣
 ↪weight
```

```
plt.imshow(gray_img, cmap='gray')
plt.axis('off')
plt.show()
```

```
(512, 512)
Time taken by OpenCV function: 0.0003559589385986328 seconds
```

[30]:

```
start = time.time()
# using for loop
gray_img = np.zeros((png_img.shape[0], png_img.shape[1]))
for i in range(png_img.shape[0]):
    for j in range(png_img.shape[1]):
        gray_img[i,j] = (png_img[i,j,0]//3 + png_img[i,j,1]//3 + png_img[i,j,2]/
 ↪/3)
end = time.time()
diff_for = end - start
print(f'Time taken by for loop: {diff_for} seconds')
plt.imshow(gray_img, cmap='gray')
plt.axis('off')
plt.show()

# time differece
speedup = diff_for/diff_lib
```

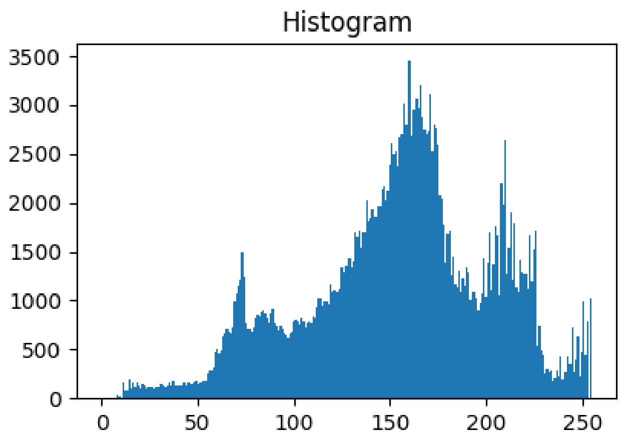
```
print(f'Speedup by using numpy is: {speedup:.2f}x')
```

```
Time taken by for loop: 0.3691413402557373 seconds
```

```
Speedup by using numpy is: 1037.03x
```

5. draw a histogram of the grayscale image

```
[31]: plt.figure(figsize=(10,3))
plt.subplot(121)
plt.title('Grayscale Image')
plt.imshow(gray_img,cmap='gray')
plt.axis('off')
# plt.show()

plt.subplot(122)
plt.title('Histogram')
plt.hist(gray_img.ravel(), bins=256, range=(0.0, 255.0), fc='k', ec='k')
plt.show()
```



6. Let X be the np matrix of the grayscale image

- obtain Y = X.T (transpose of X)
- obtain Z = Y.X (matrix multiplication of Y and X)

[32]:
```
X = gray_img
print(f"{X.shape=}")

Y = X.T
print(f"{Y.shape=}")

# multiply Y and X
Z = np.matmul(Y,X)
print(f"{Z.shape=}")
```

```
X.shape=(512, 512)
Y.shape=(512, 512)
Z.shape=(512, 512)
```

7. Obtain a portion of the grayscale image and save it in an array A

[33]:
```
P = int(X.shape[0]*.4),int(X.shape[1]*.4)
Q = int(X.shape[0]*.7),int(X.shape[1]*.7)
print(f"{P=}")
print(f"{Q=}")


# crop the image
A = gray_img[P[0]:Q[0], P[1]:Q[1]]
```

```
P=(204, 204)
Q=(358, 358)
```

8. Display the partimage A

[39]:
```
# display the originall and cropped image
plt.figure(figsize=(7,4))
plt.subplot(121)
plt.title('Original Image')
plt.imshow(gray_img, cmap='gray')
# display P and Q on the image
plt.plot([P[1], Q[1], Q[1], P[1], P[1]], [P[0], P[0], Q[0], Q[0], P[0]], 'r')
plt.text(P[1], P[0], 'P', color='b', fontsize=16)
plt.text(Q[1], Q[0], 'Q', color='b', fontsize=16)
plt.axis('off')

plt.subplot(122)
plt.title('Cropped Image A[P:Q]')
plt.imshow(A, cmap='gray')
plt.axis('off')
plt.show()
```