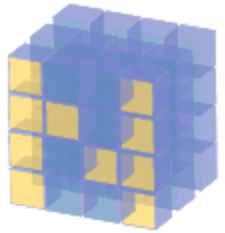
CS463/516

NumPy and matplotlib

NumPy



- NumPy is the fundamental package for scientific computing in python
- Nearly every scientist or researcher working in python uses numpy
- NumPy brings computational speed of languages like C and Fortran into python
- NumPy forms the basis for higher-level python libraries:

scikit-image
image processing in python

SciPy

pandas

How is NumPy so fast?

- Most of NumPy's low-level functions are not actually written in python
- Example: fast fourier transform (FFT)

```
from numpy.fft import fft,
```

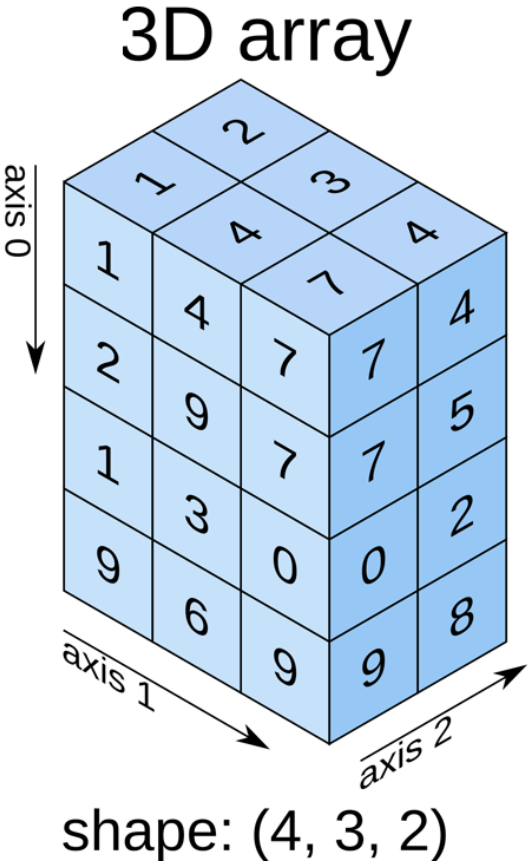
- fftpack: a package of Fortran subprograms for Fast Fourier Transform
- Another example: np.sum(array)

```
np.sum(array)
```

  - if you check out the source code for np.sum, you'll find there are no 'for' loops, instead, numpy dispatches array to another function
  - Array must be homogenously typed (all floats, all ints, etc., no mixing of float and int)
  - Uses BLAS and/or LAPACK
    - BLAS: basic linear algebra subprograms – low level C and Fortran for common linear algebra operations such as vector addition, scalar multiplication, dot products, etc.
    - LAPACK: linear algebra package written in Fortran90
- Takeaway: never use 'for' loops in python! (unless unavoidable)

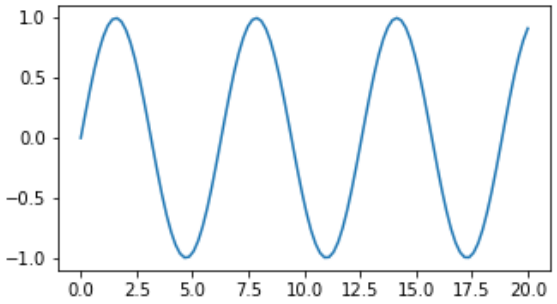
ndarray: the basic data object of numpy

- ndarray: a multidimension, homogenous array of fixed-size items
  - Associated to a data-type object describing format of each element (integer, float, etc.)
- ndarray has several important attributes:
  - `dtype` – data type of the array's elements
  - `shape` – tuple of array dimensions
  - Others: `imag`, `real`, `size`, `ndim`, `nbytes`, etc. (check link below for full list)
- And several important methods:
  - `min`, `max` – return min or max value
  - `argmin`, `argmax` – return indices of min or max values
  - `reshape` – change shape of array
  - `sort` – sort in ascending/descending order along a dimension
  - `ravel` – flatten the array (make it 1-dimensional)
  - `astype` – cast array items to another type (e.g. float to int)
  - Others: `mean`, `std`, `cumsum`, etc (check link below for full list)
  - can also use `np.min(array)`, `np.reshape(array)`, etc., instead



https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html

Linear spaces and grids

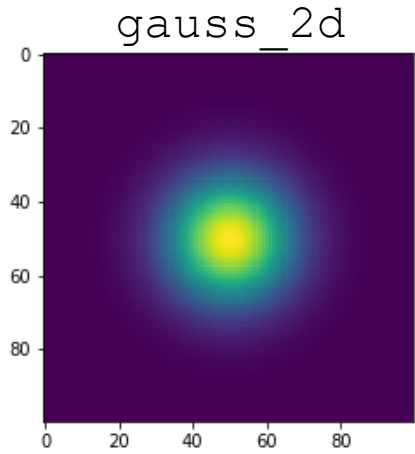


- Two very useful numpy functions:
- **1) np.linspace()** – return evenly spaced numbers over a specified interval
- Example: create a sin wave: `np.sin(np.linspace(0,20,100))`
  - Applies the sin function to 100 evenly spaced points from 0 to 20
  - To plot, use matplotlibs 'plot' function and give another linspace as x-axis:

```
plt.plot(np.linspace(0,20,100),np.sin(np.linspace(0,20,100)))
```

- **2) np.mgrid()** – returns a dense multidimensional meshgrid
  - meshgrid: n-dimensional coordinate array for use in vectorized evaluation of scalar/vector fields
- Example: create a 2d gaussian: $G_\sigma(x,y) = \frac{1}{2\pi\sigma^2} e^{\left(-\frac{x^2+y^2}{2\sigma^2}\right)}$

```
sz_x = 100; sz_y = 100;
[X, Y] = np.mgrid[0:sz_x, 0:sz_y]
xpr = X - sz_x // 2
ypr = Y - sz_y // 2
sigma = 12
gauss_2d = np.exp(-((xpr**2+ypr**2)/(2*sigma**2)))/(2*np.pi*sigma**2)
```



Indexing and array initialization in NumPy

```
In [46]: nii.shape
Out[46]: (274, 384, 384)
```

- ndarrays can be initialized in many ways:
  - An empty array of zeros or ones
  - An array of random numbers
  - Copying of a pre-existing array

```
nii = nib.load('C:/shared/t1.nii')
zeros_img = np.zeros(nii.shape)
ones_img = np.ones(nii.shape)
tens_img = np.ones(nii.shape)*10

new_3d_zeros = np.zeros([100,100,100])
new_3d_rnd = np.random.rand(100,100,100)
```

- Once we have array, many ways to access elements
- Colon ':' operator used to 'slice' the array
  - Plot a single slice:
    - coronal (a) `plt.imshow(nii.get_data()[:,200,:])`
    - axial (b) `plt.imshow(nii.get_data()[:,:,200])`
  - Plot every 8th column/row from a slice (c)

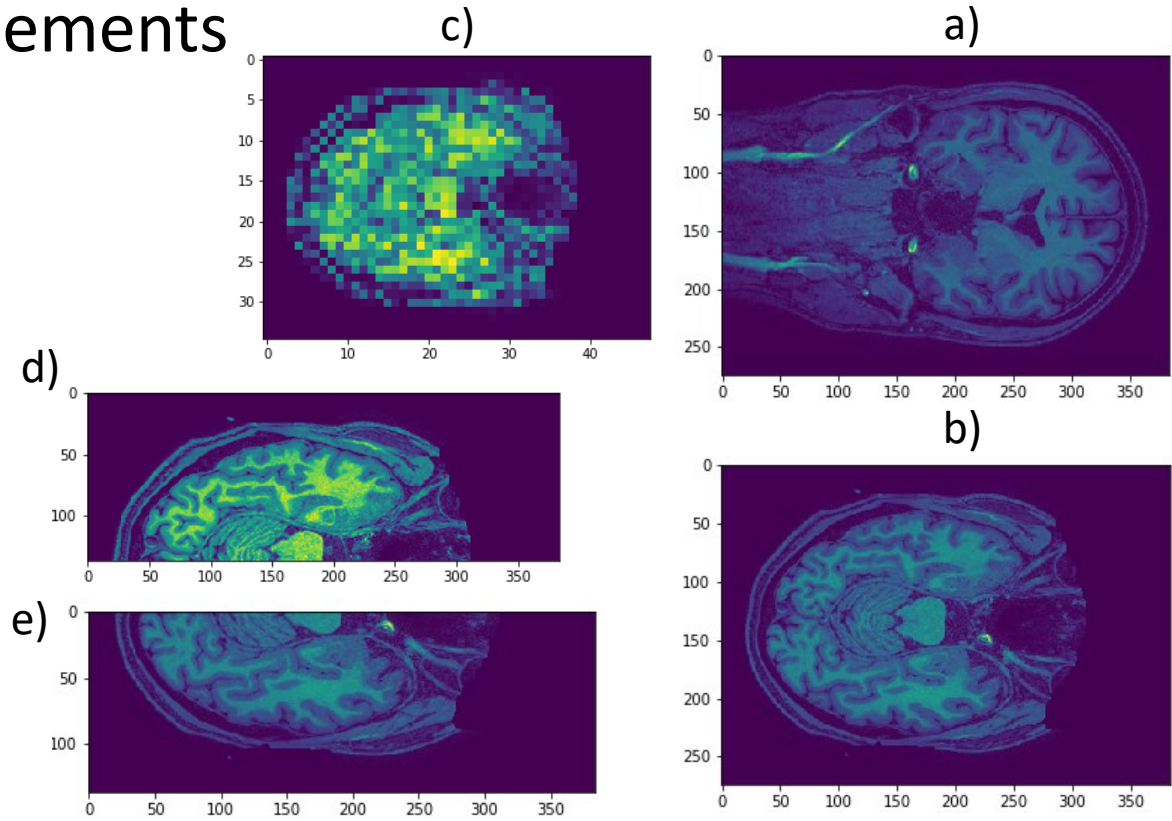
```
plt.imshow(nii.get_data()[::8,::8,200])
```

  - Plot the first half of a slice (d)

```
plt.imshow(nii.get_data()[:-nii.shape[0]//2,:,200])
```

  - Plot the 2nd half of a slice (e)

```
plt.imshow(nii.get_data()[nii.shape[0]//2::,:,200])
```



matplotlib: visualization with python

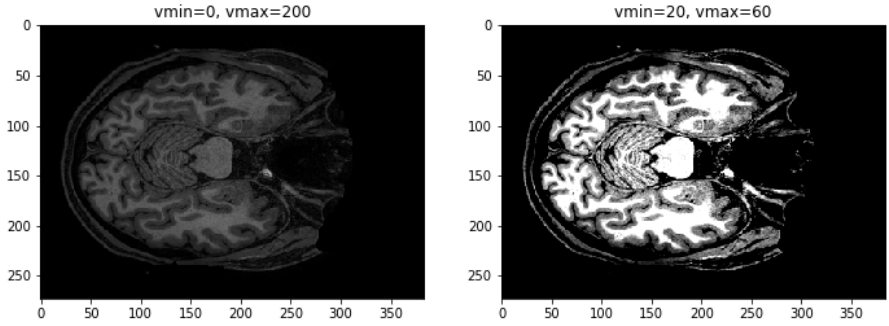
- In medical imaging (or any image processing), very important to be able to visualize the results of your algorithms and convey results to other researchers
- Typically, a basic image display or time series (plot) is sufficient
- Matplotlib: library for creating static, animated, and interactive visualizations in python
- *Interactive figures* – scroll through data, examine values at specific data points, etc
- *Publication-quality plots* – output figures in high quality vector graphics formats such as .eps or .svg
- *Full customization* – modify axis labels, axis tick values, colormaps, legends, colorbars, etc.

pyplot: imshow and plot

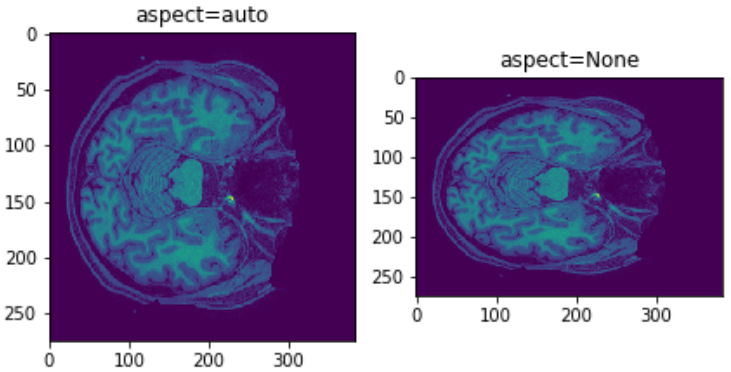
```
import matplotlib.pyplot as plt
```

- imshow and plot from the pyplot package is all you need for this course
  - Allows for efficient visualization of time series/scatter plots (plot) and images (imshow)
- Pyplot: provides a MATLAB-like plotting framework
- `plt.imshow`: plots an (n,m) or (n,m,3) or (n,m,4) array as an image (grayscale, rgb, rgba respectively)
  - Important options
    - cmap – select the colormap (a)
    - vmin, vmax – 'clamp' the intensity within a certain range (b)
    - aspect – stretch image to fit plot, or keep original aspect ratio (c)

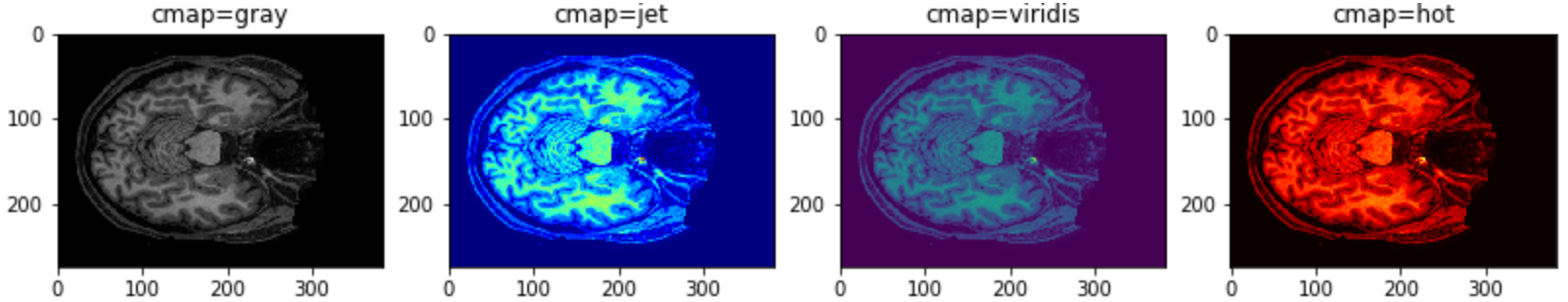
b)



c)



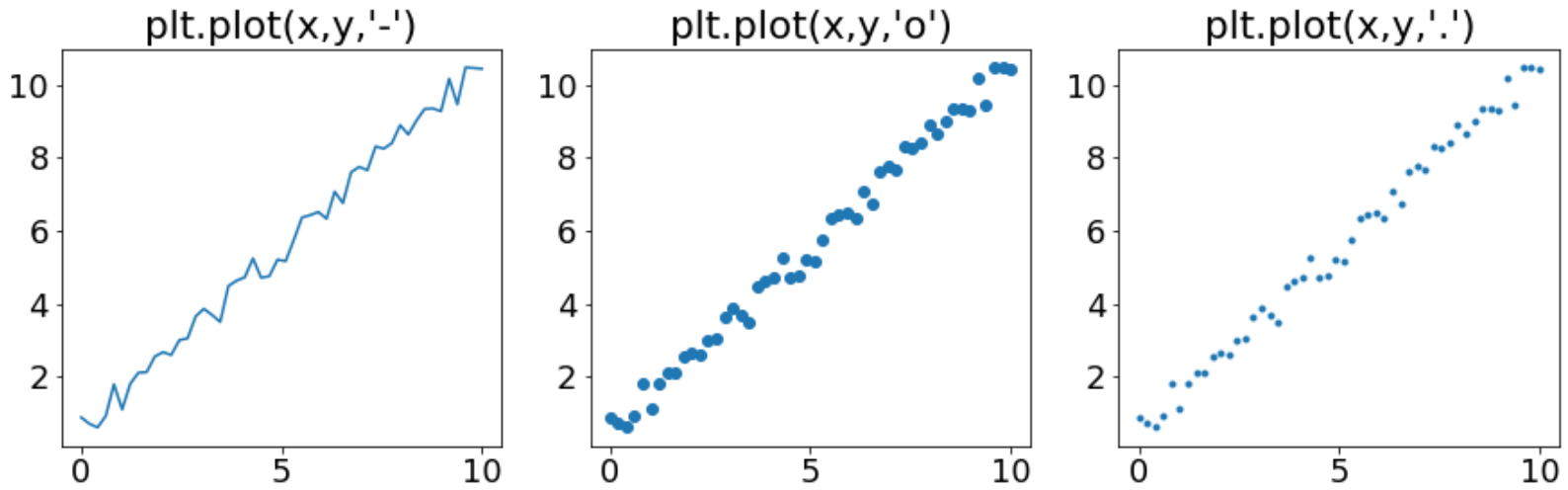
a)



plt.plot()

- Probably the most widely used matplotlib function
- Mainly for plotting scatter plots and time series
- Example: create a random scatter plot and plot in different styles

```
x = np.linspace(0,10,num=50)
y = np.random.rand(50) + x
plt.subplot(1,3,1); plt.plot(x,y,'-'); plt.title('plt.plot(x,y,\'-\')')
plt.subplot(1,3,2); plt.plot(x,y,'o'); plt.title('plt.plot(x,y,\'o\')')
plt.subplot(1,3,3); plt.plot(x,y,'.'); plt.title('plt.plot(x,y,\'.\')')
```



Numpy/matplotlib demo (see video)

- Linspace
- Ndgrid
- Sorting
- Flattening
- Array initialization
- Indexing
- broadcasting
- Matplotlib.pyplot

```
In [86]: arr1 = np.zeros([100,100])
    ...: arr2 = np.ones([100,1])
    ...:

In [87]: arr1+arr2
Out[87]:
array([[1., 1., 1., ..., 1., 1., 1.],
       [1., 1., 1., ..., 1., 1., 1.],
       [1., 1., 1., ..., 1., 1., 1.],
       ...,
       [1., 1., 1., ..., 1., 1., 1.],
       [1., 1., 1., ..., 1., 1., 1.],
       [1., 1., 1., ..., 1., 1., 1.]])

In [88]: (arr1+arr2).shape
Out[88]: (100, 100)

In [89]: arr1 = np.zeros([100,100])
    ...: arr2 = np.ones([99,1])
    ...:
    ...:

In [90]: arr1+arr2
Traceback (most recent call last):

  File "<ipython-input-90-e489ba1ad4d1>", line 1, in <module>
    arr1+arr2

ValueError: operands could not be broadcast together with shapes (100,100) (99,1)
```