Programming for Data Science: Numpy and Scipy

Instructor: Jakramate Bootkrajang

Based on material from https://scipy-lectures.org/

Outline

- Numpy
- Scipy

NumPy

- extension package to Python for multi-dimensional arrays
- closer to hardware (efficiency)
- designed for scientific computation (convenience)
- Also known as array oriented computing

Importing Numpy

- Using **import** keyword
- Package can be renamed using **as** keyword

```
>>> import numpy as np
>>> a = np.array([0, 1, 2, 3])
>>> a
array([0, 1, 2, 3])
```

Array can contain

- values of an experiment/simulation at discrete time steps
- signal recorded by a measurement device, e.g. sound wave
- pixels of an image, grey-level or colour
- 3-D data measured at different X-Y-Z positions, e.g. MRI scan

Creating arrays

- Using array() function which takes as input a list

1-D:

```
>>> a = np.array([0, 1, 2, 3])
>>> a
array([0, 1, 2, 3])
>>> a.ndim
1
>>> a.shape
(4,)
>>> len(a)
4
```

Creating array (cont.)

2-D, 3-D, ...:

```
>>> b = np.array([[0, 1, 2], [3, 4, 5]])    # 2 x 3 array
>>> b
array([[0, 1, 2],
       [3, 4, 5]])
>>> b.ndim
2
>>> b.shape
(2, 3)
>>> len(b)     # returns the size of the first dimension
2

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

       [[3],
        [4]]])
>>> c.shape
(2, 2, 1)
```

Functions for creating arrays

- In practice, we rarely enter items one by one...
- We can use arange() to create evenly spaced array

Evenly spaced:

```
>>> a = np.arange(10) # 0 .. n-1  (!)
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> b = np.arange(1, 9, 2) # start, end (exclusive), step
>>> b
array([1, 3, 5, 7])
```

Functions for creating arrays (cont.)

- Or create an array by number of points

```
>>> c = np.linspace(0, 1, 6)   # start, end, num-points
>>> c
array([ 0. ,  0.2,  0.4,  0.6,  0.8,  1. ])
>>> d = np.linspace(0, 1, 5, endpoint=False)
>>> d
array([ 0. ,  0.2,  0.4,  0.6,  0.8])
```

Array reversing

- The usual python idiom for reversing a sequence is supported:

```
>>> a[::-1]
array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
```

Common arrays

```
>>> a = np.ones((3, 3))  # reminder: (3, 3) is a tuple
>>> a
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
>>> b = np.zeros((2, 2))
>>> b
array([[ 0.,  0.],
       [ 0.,  0.]])
>>> c = np.eye(3)
>>> c
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
>>> d = np.diag(np.array([1, 2, 3, 4]))
>>> d
array([[1, 0, 0, 0],
       [0, 2, 0, 0],
       [0, 0, 3, 0],
       [0, 0, 0, 4]])
```

Array with random entries

- Using rand() in random sub-package

```
>>> a = np.random.rand(4)        # uniform in [0, 1]
>>> a
array([ 0.95799151,  0.14222247,  0.08777354,  0.51887998])

>>> b = np.random.randn(4)       # Gaussian
>>> b
array([ 0.37544699, -0.11425369, -0.47616538,  1.79664113])

>>> np.random.seed(1234)         # Setting the random seed
```

Array indexing

- The items of an array can be accessed and assigned to the same way as other Python sequences (e.g. lists):

```
>>> a = np.arange(10)
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> a[0], a[2], a[-1]
(0, 2, 9)
```

Multi-dimensional array indexing

- For multidimensional arrays, indexes are tuples of integers:

```
>>> a = np.diag(np.arange(3))
>>> a
array([[0, 0, 0],
       [0, 1, 0],
       [0, 0, 2]])
>>> a[1, 1]
1
>>> a[2, 1] = 10 # third line, second column
>>> a
array([[ 0,  0,  0],
       [ 0,  1,  0],
       [ 0, 10,  2]])
>>> a[1]
array([0, 1, 0])
```

Array slicing

- Arrays, like other Python sequences can also be sliced

```
>>> a = np.arange(10)
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> a[2:9:3] # [start:end:step]
array([2, 5, 8])
```

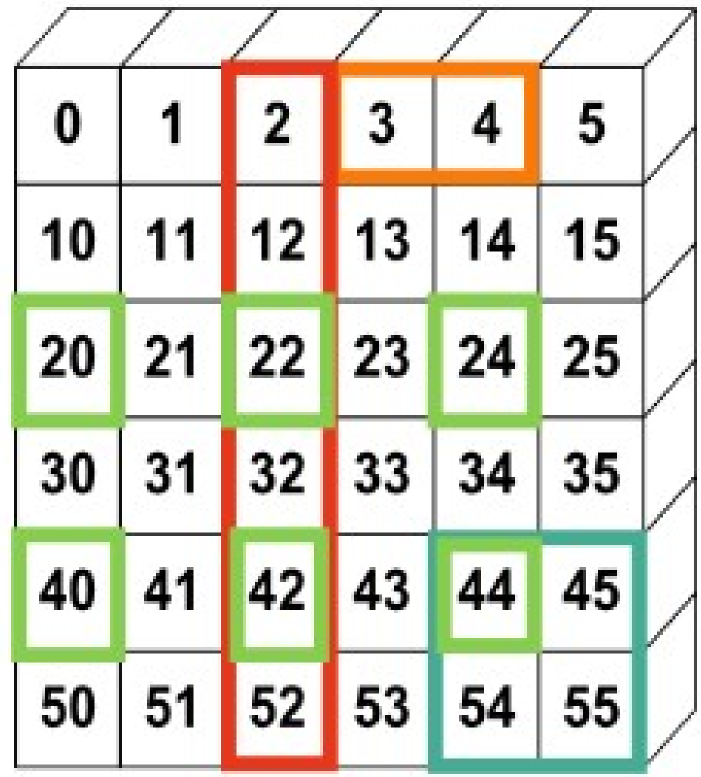
Visual summary

```
>>> a[0,3:5]
array([3,4])

>>> a[4:,4:]
array([[44, 45],
       [54, 55]])

>>> a[:,2]
array([2,12,22,32,42,52])

>>> a[2::2,::2]
array([[20,22,24]
       [40,42,44]])
```



Fancy indexing

- NumPy arrays can be indexed with slices, but also with boolean or integer arrays (masks). This method is called fancy indexing

```
>>> np.random.seed(3)
>>> a = np.random.randint(0, 21, 15)
>>> a
array([10,  3,  8,  0, 19, 10, 11,  9, 10,  6,  0, 20, 12,  7, 14])
>>> (a % 3 == 0)
array([False,  True, False,  True, False, False, False,  True, False,
        True,  True, False,  True, False, False], dtype=bool)
>>> mask = (a % 3 == 0)
>>> extract_from_a = a[mask] # or,  a[a%3==0]
>>> extract_from_a           # extract a sub-array with the mask
array([ 3,  0,  9,  6,  0, 12])
```

Assigning new values with a mask

```
>>> a[a % 3 == 0] = -1
>>> a
array([10, -1,  8, -1, 19, 10, 11, -1, 10, -1, -1, 20, -1,  7, 14])
```

Fancy indexing with list of integers

- Indexing can be done with an array of integers, where the same index is repeated several time:

```
>>> a = np.arange(0, 100, 10)
>>> a
array([ 0, 10, 20, 30, 40, 50, 60, 70, 80, 90])
```

```
>>> a[[9, 7]] = -100
>>> a
array([   0,   10,   20,   30,   40,   50,   60, -100,   80, -100])
```

Operations on array

With scalars:

```
>>> a = np.array([1, 2, 3, 4])
>>> a + 1
array([2, 3, 4, 5])
>>> 2**a
array([ 2,  4,  8, 16])
```

All arithmetic operates elementwise:

```
>>> b = np.ones(4) + 1
>>> a - b
array([-1.,  0.,  1.,  2.])
>>> a * b
array([ 2.,  4.,  6.,  8.])

>>> j = np.arange(5)
>>> 2**(j + 1) - j
array([ 2,  3,  6, 13, 28])
```

Note !

- Array multiplication is not matrix multiplication

```
>>> c = np.ones((3, 3))
>>> c * c                       # NOT matrix multiplication!
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
```

Note: Matrix multiplication:

```
>>> c.dot(c)
array([[ 3.,  3.,  3.],
       [ 3.,  3.,  3.],
       [ 3.,  3.,  3.]])
```

Array comparison

Comparisons:

```
>>> a = np.array([1, 2, 3, 4])
>>> b = np.array([4, 2, 2, 4])
>>> a == b
array([False,  True, False,  True], dtype=bool)
>>> a > b
array([False, False,  True, False], dtype=bool)
```

Array-wise comparisons:

```
>>> a = np.array([1, 2, 3, 4])
>>> b = np.array([4, 2, 2, 4])
>>> c = np.array([1, 2, 3, 4])
>>> np.array_equal(a, b)
False
>>> np.array_equal(a, c)
True
```

Logical operations

Logical operations:

```
>>> a = np.array([1, 1, 0, 0], dtype=bool)
>>> b = np.array([1, 0, 1, 0], dtype=bool)
>>> np.logical_or(a, b)
array([ True,  True,  True, False], dtype=bool)
>>> np.logical_and(a, b)
array([ True, False, False, False], dtype=bool)
```

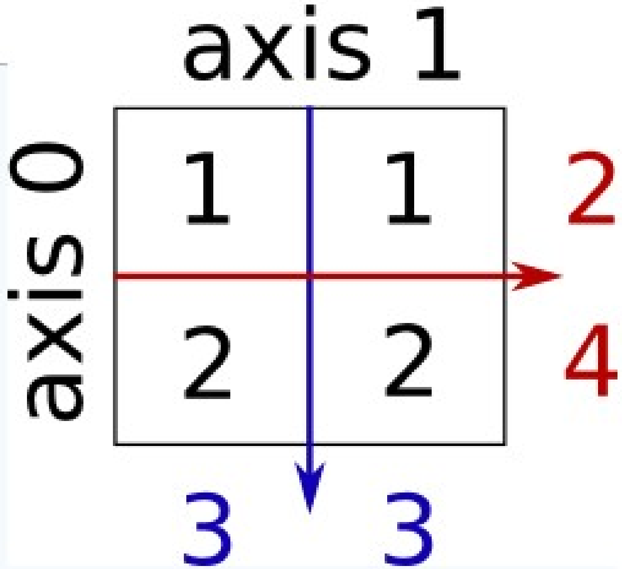
Others functions

```
>>> a = np.arange(5)
>>> np.sin(a)
array([ 0.        ,  0.84147098,  0.90929743,  0.14112001, -0.7568025 ])
>>> np.log(a)
array([       -inf,  0.        ,  0.69314718,  1.09861229,  1.38629436])
>>> np.exp(a)
array([  1.        ,   2.71828183,   7.3890561 ,  20.08553692,  54.59815003])
```

Array reductions

Sum by rows and by columns:

```
>>> x = np.array([[1, 1], [2, 2]])
>>> x
array([[1, 1],
       [2, 2]])
>>> x.sum(axis=0)   # columns (first dimension)
array([3, 3])
>>> x[:, 0].sum(), x[:, 1].sum()
(3, 3)
>>> x.sum(axis=1)   # rows (second dimension)
array([2, 4])
>>> x[0, :].sum(), x[1, :].sum()
(2, 4)
```



Other reduction

Extrema:

```
>>> x = np.array([1, 3, 2])
>>> x.min()
1
>>> x.max()
3

>>> x.argmin()  # index of minimum
0
>>> x.argmax()  # index of maximum
1
```

Logical operations:

```
>>> np.all([True, True, False])
False
>>> np.any([True, True, False])
True
```

Statistics

```
>>> x = np.array([1, 2, 3, 1])
>>> y = np.array([[1, 2, 3], [5, 6, 1]])
>>> x.mean()
1.75
>>> np.median(x)
1.5
>>> np.median(y, axis=-1) # last axis
array([ 2.,  5.])

>>> x.std()             # full population standard dev.
0.82915619758884995
```

Sorting data

Sorting along an axis:

```
>>> a = np.array([[4, 3, 5], [1, 2, 1]])
>>> b = np.sort(a, axis=1)
>>> b
array([[3, 4, 5],
       [1, 1, 2]])
```

Scipy

- high-level scientific computing
- scipy package contains various toolboxes dedicated to common issues in scientific computing
- Its different submodules correspond to different applications,
  - interpolation,
  - integration,
  - optimization,
  - image processing, etc.

Scipy contents

scipy is composed of task-specific sub-modules:

`scipy.cluster`	Vector quantization / Kmeans
`scipy.constants`	Physical and mathematical constants
`scipy.fftpack`	Fourier transform
`scipy.integrate`	Integration routines
`scipy.interpolate`	Interpolation
`scipy.io`	Data input and output
`scipy.linalg`	Linear algebra routines
`scipy.ndimage`	n-dimensional image package
`scipy.odr`	Orthogonal distance regression
`scipy.optimize`	Optimization
`scipy.signal`	Signal processing
`scipy.sparse`	Sparse matrices
`scipy.spatial`	Spatial data structures and algorithms
`scipy.special`	Any special mathematical functions
`scipy.stats`	Statistics

Scipy and Numpy

- Scipy sub-modules are all depend on numpy
- Standard way of importing numpy and scipy is

```
>>> import numpy as np
>>> from scipy import stats  # same for other sub-modules
```

Scipy.linalg

- Provides standard linear algebra operations
  - The `scipy.linalg.det()` function computes the determinant of a square matrix:

```
>>> from scipy import linalg
>>> arr = np.array([[1, 2],
...                 [3, 4]])
>>> linalg.det(arr)
-2.0
>>> arr = np.array([[3, 2],
...                 [6, 4]])
>>> linalg.det(arr)
0.0
>>> linalg.det(np.ones((3, 4)))
Traceback (most recent call last):
...
ValueError: expected square matrix
```

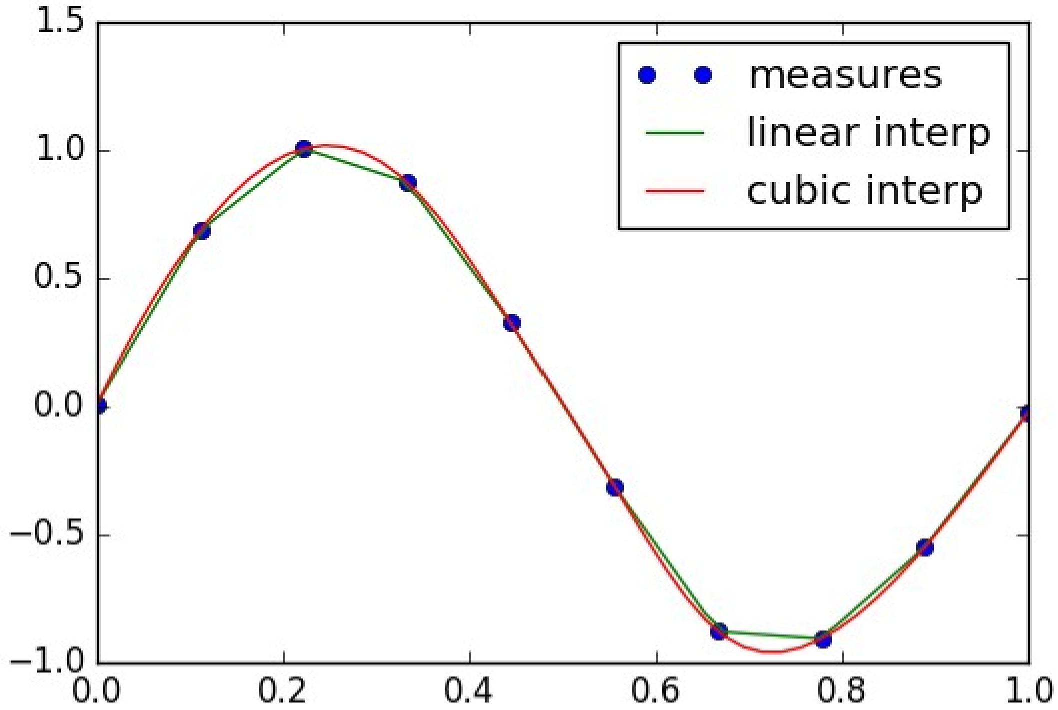
Scipy.linalg (cont.)

- The `scipy.linalg.inv()` function computes the inverse of a square matrix:

```
>>> arr = np.array([[1, 2],
...                 [3, 4]])
>>> iarr = linalg.inv(arr)
>>> iarr
array([[-2. ,  1. ],
       [ 1.5, -0.5]])
>>> np.allclose(np.dot(arr, iarr), np.eye(2))
True
```

scipy.interpolate

- ***scipy.interpolate*** is useful for fitting a function from experimental data and thus evaluating points where no measure exists.



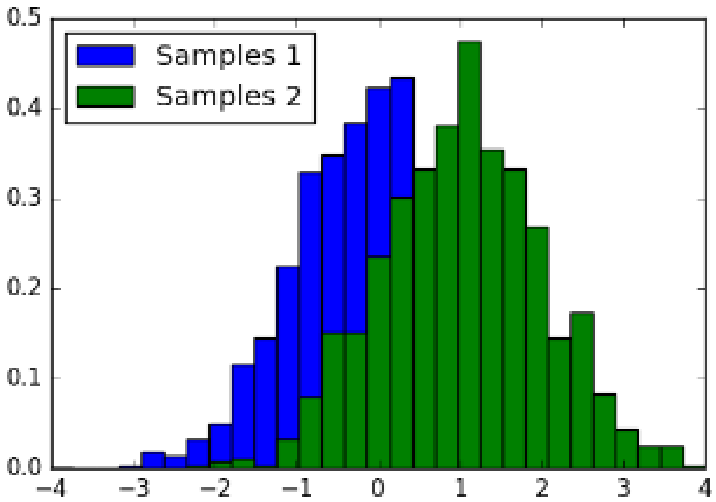
scipy.stats

- The module scipy.stats contains statistical tools and probabilistic descriptions of random processes
- For example
  - Constructing histogram
  - Calculating mean, median, percentile
  - Performing statistical test

Performing t-test

- decide whether the means of two sets of observations are significantly different

```
>>> a = np.random.normal(0, 1, size=
      100)
>>> b = np.random.normal(1, 1, size=10)
>>> stats.ttest_ind(a, b)
(array(-3.177574054...),
      0.0019370639...)
```



- ttest_ind() returns *significant of difference* and *p-value*

Reading list

- https://scipy-lectures.org/intro/numpy/index.html
- https://scipy-lectures.org/intro/scipy.html