PennState World Campus

DAAN862: Analytics Programming in Python

Lesson 2: Numeric Analysis with Numpy

Lesson 2: Objectives and Overview (1 of 6)

Lesson 2: Numeric Analysis with Numpy

This lesson introduces one of the basic Python packages: Numpy, which is a fundamental package required for high-performance scientific computing and data analysis.

At the end of this lesson, students will be able to:

- Recognize how to generate arrays by use of Numpy functions
- Select items or sub-arrays from a Numpy array
- Utilize Numpy array methods and functions

By the end of this lesson, please complete all readings and assignments found in the Lesson 2 Course Schedule.

Lesson 2.1: Arrays (2 of 6)

Lesson 2.1: Arrays

(click the titles to learn more)

Arrays

Numpy (Numeric Python), is the core package required for scientific computing and data analysis (scipy, pandas, sklearn, etc). In addition, it is a foundation for nearly all of the higher-level packages in Python. It provides the following:

- ndarray, a multidimensional array which provides fast adn space-efficient vectorized arithmetic operations
- Fast standard and mathematical functions which execute on entire arrays of data without using loops
- Linear algebra and random generation

Example

Importing the Numpy Package

```
In [1]: import numpy as np
```

Fig 2.1(click to enlarge)

np is the conventional short name for numpy. You can choose your own name to import numpy.

Numpy ndarray is:

A table of elements with the same type (typically elements should be numbers)

Indexed by a tuple of integers

It's dimensions are called axes

Array Creation

The easiest way to create an array is to use the array function. This accepts any sequence-like object and produces a new NumPy array containing the passed data. NumPy also provides some standard array creation functions, as shown in table 2.1.1.

Table 2.1.1 Array Creation Function

Function	Descriptions
numpy.array	Create an array.
numpy.zeros	Return a new array of given shape and type, filled with zeros.
numpy.zeros_like	Return an array of zeros with the same shape and type as a given array.
numpy.ones	Return a new array of given shape and type, filled with ones.
numpy.ones_like	Return an array of ones with the same shape and type as a given array.

Function	Descriptions
numpy.empty	Return a new array of given shape and type, without initializing entries.
numpy.empty_like	Return a new array with the same shape and type as a given array.
numpy.arrange	Return evenly spaced values within a given interval.
numpy.linspace	Return evenly spaced numbers over a specified interval.
numpy.random.rand	Create an array of the given shape and populate it with random samples from a uniform distribution over [0, 1).
numpy.random.randn	Return a sample (or samples) from the “standard normal” distribution.
numpy.fromfucntion	Construct an array by executing a function over each coordinate.
numpy.fromfile	Construct an array from data in a text or binary file.

Example

Click the dropdown to see examples of how to use these functions to create *ndarry*

```
In [2]: np.array(range(10))         # convert a list to array
Out[2]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [3]: A = np.array([[1,2], [3,4]]) # convert a list of lists to 2-D array.

In [4]: print(A)
[[1 2]
 [3 4]]

In [5]: np.zeros( (3,4) )           #Create a 3x4 zero array
Out[5]:
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])

In [6]: np.zeros_like(A)            # Create an array of 0s with the same shape as A
Out[6]:
array([[0, 0],
       [0, 0]])

In [7]: np.ones((2,3,4), dtype=np.int16)   # create a 2x3x4 array with 1s
Out[7]:
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]]], dtype=int16)

In [8]: np.ones_like(A)             # Create an array of 1s with the same shape as A
Out[8]:
array([[1, 1],
       [1, 1]])

In [9]: np.empty((2,3))             # Create an 2x3 empty array
Out[9]:
array([[0., 0., 0.],
       [0., 0., 0.]])

In [10]: np.empty_like(A)            # Create an empty array with the same shape as A
Out[10]:
array([[ 543236212, 2004116846],
       [ 543912559, 1701470831]])

In [11]: np.arange(10, 30, 5)        # create an array with evenly spaced values
Out[11]: array([10, 15, 20, 25])

In [12]: np.arange(0, 2, 0.3)
Out[12]: array([0. , 0.3, 0.6, 0.9, 1.2, 1.5, 1.8])

In [13]: np.linspace(0, 2, 9)        # create an array which divides the interval evenly
Out[13]: array([0.  , 0.25, 0.5 , 0.75, 1.  , 1.25, 1.5 , 1.75, 2.  ])
```

Fig 2.2 (A) (click to enlarge)

```
In [14]: np.random.seed(123)         # set the random seed

In [15]: np.random.rand(3,2)
Out[15]:
array([[0.69646919, 0.28613933],
       [0.22685145, 0.55131477],
       [0.71946897, 0.42310646]])

In [16]: np.random.randn(3, 4)
Out[16]:
array([[-2.42667924, -0.42891263,  1.26593626, -0.8667404 ],
       [-0.67888615, -0.09470897,  1.49138963, -0.638902  ],
       [-0.44398196, -0.43435128,  2.20593008,  2.18678609]])
```

Fig 2.2 (B) (click to enlarge)

Please click here (https://docs.scipy.org/doc/numpy/reference/routines.array-creation.html) for details of these array creation function and more array creatation methods.

The important attributes of a ndarray object are:

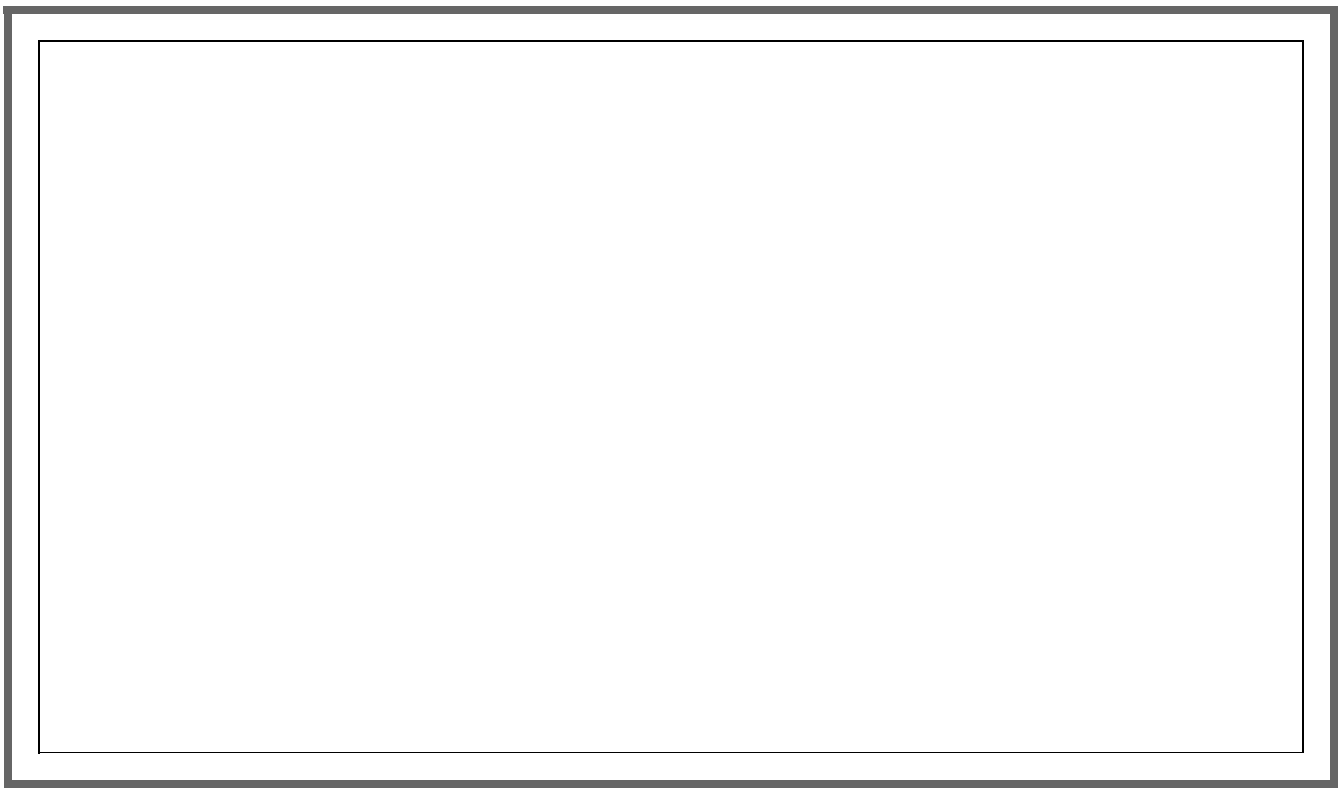
Table 2.1.2

Methods	Description
ndarray.ndim	the number of axes (dimensions) of the array
ndarray.shape	the dimensions of the array. This is a tuple of integers.
ndarray.size	the total number of elements of the array.
ndarray.dtype	the type of the elements in the array.

Example

Click the dropdown to view examples of methods in Table 2.1.2

Fig 2.3
(click to
enlarge)



Lesson 2.2: Array Indexing

(click the tabs to learn more)

Indexing One-Dimensional Array:

One-Dimensional arrays can be indexed, sliced, and iterated over like lists.

Example

Click the dropdown to view an example of One-Dimensional Array

```
In [24]: a = np.arange(6)**2

In [25]: a
Out[25]: array([ 0,  1,  4,  9, 16, 25], dtype=int32)

In [26]: a[2]                    # Select the 3rd element
Out[26]: 4

In [27]: a[2:5]                  # Select the 3rd - 5th elements
Out[27]: array([ 4,  9, 16], dtype=int32)

In [28]: a[:6:2] = -100  # equivalent to a[0:6:2] = -1000; from start to position 6, excl
every 2nd element to -1000

In [29]: a[::-1]                 # reverse the array
Out[29]: array([  25, -100,    9, -100,    1, -100], dtype=int32)
```

Fig 2.4 (click to enlarge)

Indexing Two-Dimensional Array:

Multidimensional arrays can have one index per axis. These indices are given in a tuple separated by commas:

Example

Click the dropdown to view an example of a Multidimensional array

```
In [30]: def f(x, y):
    ...:     return 10*x + y

In [31]: b = np.fromfunction(f, (5, 4), dtype = int) # create an array from the functio

In [32]: b
Out[32]:
array([[ 0,  1,  2,  3],
       [10, 11, 12, 13],
       [20, 21, 22, 23],
       [30, 31, 32, 33],
       [40, 41, 42, 43]])

In [33]: b[2,3]                 # select the item at 2rd row and 3nd column
Out[33]: 23

In [34]: b[0:3, 1]              # select row 0-2 in the 1st column
Out[34]: array([ 1, 11, 21])

In [35]: b[ : ,1]               # Select the 2nd column
Out[35]: array([ 1, 11, 21, 31, 41])

In [36]: b[1:3, : ]             # all columns in the second and third row of b
Out[36]:
array([[10, 11, 12, 13],
       [20, 21, 22, 23]])
```

Fig 2.5 (click to enlarge)

When fewer indices are provided than the number of axes, the missing indices are considered as complete slices as shown in Figure 2.6:

```
In [37]: b[-1]                  # the last row. Equivalent to b[-1,:]
Out[37]: array([40, 41, 42, 43])
```

Fig 2.6 (click to enlarge)

Boolean Indexing:

A random array called *data*, and another array called *name* which contains duplicates.

```
In [38]: names = np.array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'])

In [39]: names
Out[39]: array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'], dtype='<U4')

In [40]: data = np.random.randn(7, 4)

In [41]: data
Out[41]:
array([[ 1.0040539 ,  0.3861864 ,  0.73736858,  1.49073203],
       [-0.93583387,  1.17582904, -1.25388067, -0.6377515 ],
       [ 0.9071052 , -1.4286807 , -0.14006872, -0.8617549 ],
       [-0.25561937, -2.79858911, -1.7715331 , -0.69987723],
       [ 0.92746243, -0.17363568,  0.00284592,  0.68822271],
       [-0.87953634,  0.28362732, -0.80536652, -1.72766949],
       [-0.39089979,  0.57380586,  0.33858905, -0.01183049]])
```

Fig 2.7 (click to enlarge)

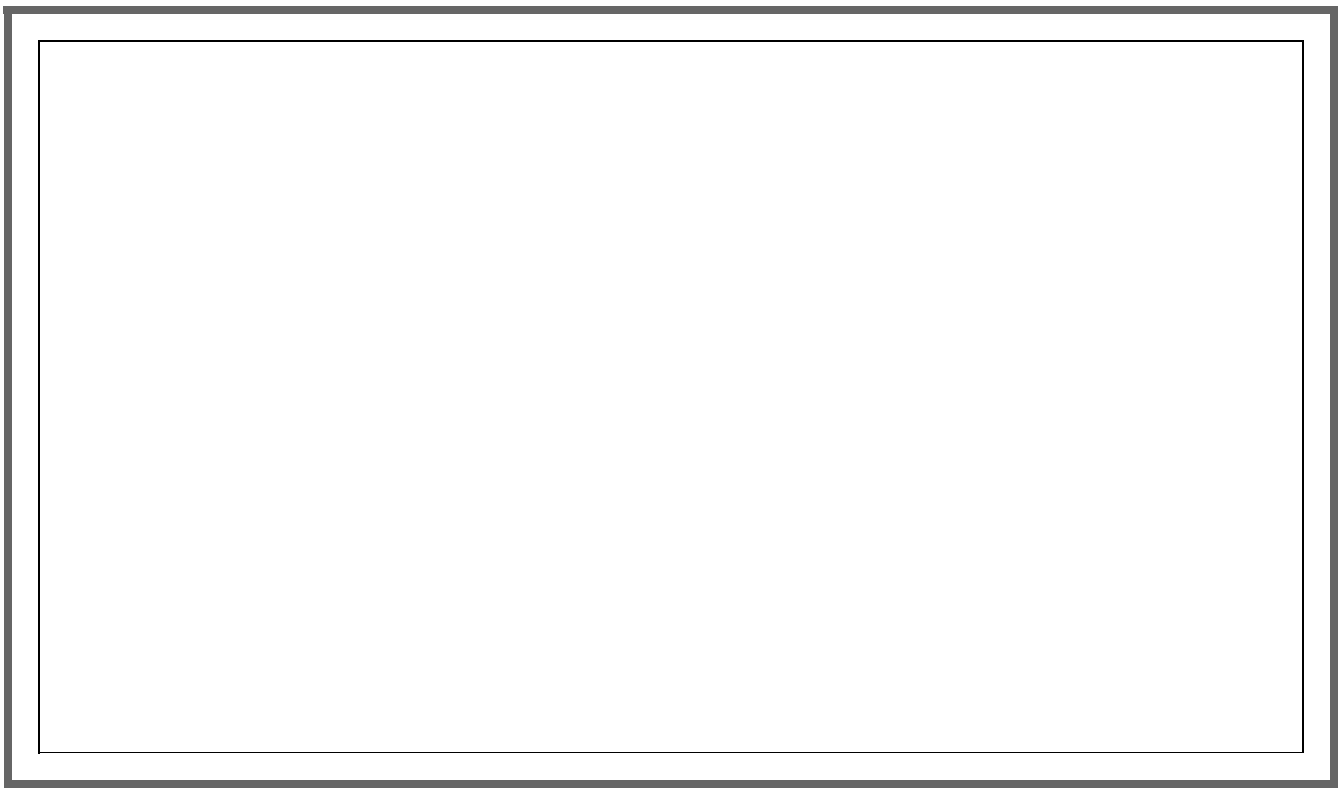
Data can be indexed by Boolean array:

```
In [42]: names == 'Bob'          # Boolean array
Out[42]: array([ True, False, False,  True, False, False, False])

In [43]: data[names == 'Bob']    # select all columns with rows when names == 'Bob'
Out[43]:
array([[ 1.0040539 ,  0.3861864 ,  0.73736858,  1.49073203],
       [-0.25561937, -2.79858911, -1.7715331 , -0.69987723]])

In [44]: data[names != 'Bob', 3] # inverse selection
Out[44]: array([-0.6377515 , -0.8617549 ,  0.68822271, -1.72766949, -0.01183049])
```

Fig 2.8 (click to enlarge)



Lesson 2.3: Array Operation

(click the titles to learn more)

Operations Between Array's and Scalars

Array's are important because they enable you to express batch operations on data without writing any for loops. This is called *vectorization*. Any arithmetic operations between equal-size arrays applies the operation element wise.

Example

Click the dropdown to view basic math operations on arrays

```
In [45]: a = np.array( [12, 43, 4, 89, 65] )

In [46]: b = np.arange( 5 )

In [47]: c = a + b              # assign elementwise sum of a and b to c

In [48]: c
Out[48]: array([12, 44,  6, 92, 69])

In [49]: a * b
Out[49]: array([  0,  43,   8, 267, 260])

In [50]: b**3
Out[50]: array([ 0,  1,  8, 27, 64], dtype=int32)

In [51]: 2 * np.exp(b)
Out[51]:
array([  2.        ,   5.43656366,  14.7781122 ,  40.17107385,
       109.19630007])

In [52]: a >= 20
Out[52]: array([False,  True, False,  True,  True])
```

Fig 2.9 (click to enlarge)

Statistic Methods

Numpy provides a set of mathematical methods to ndarray for statistic analysis on the entire array or along an axis. Aggregations like sum, mean, and standard deviation (std) can either be used by calling the array method or using the top level Numpy functions. Refer to Table 2.3.1 below for the list of statistic methods:

Table 2.3.1: The Basic Methods

Method	Description
sum	Sum of all elements in the array or along an axis
mean	Arithmetic mean
std, var	Standard deviation and variance
min,max	Minimum and Maximum
argmin,argmax	Indices of minimum and maximum elements
cumsum	Cumulative sum of elements starting from 0
cumprod	Cumulative product of elements starting from 1

Example

Click the dropdown to view examples of how to use statistic methods

```
In [53]: a = np.random.random((2,3))

In [54]: a
Out[54]:
array([[0.51948512, 0.61289453, 0.12062867],
       [0.8263408 , 0.60306013, 0.54506801]])

In [55]: a.sum()                # sum of all items
Out[55]: 3.227477246397326

In [56]: a.sum(axis = 0)        # column sum
Out[56]: array([1.34582592, 1.21595465, 0.66569667])

In [57]: a.sum(axis = 1)        # row sum
Out[57]: array([1.25300831, 1.97446894])

In [58]: np.sum(a)              # equivalent to a.sum()
Out[58]: 3.227477246397326

In [59]: np.sum(a, axis = 0)    # equivalent to a.sum(axis = 0)
Out[59]: array([1.34582592, 1.21595465, 0.66569667])

In [60]: a.min()                # minimum of a
Out[60]: 0.12062866599032374

In [61]: a.min(axis = 0)        # minimun for each column
Out[61]: array([0.51948512, 0.60306013, 0.12062867])

In [62]: np.min(a)              # equivalent to a.min()
Out[62]: 0.12062866599032374

In [63]: a.max()                # maximum of a
Out[63]: 0.8263408005068332

In [64]: np.max(a)              # equivalent to a.max()
Out[64]: 0.8263408005068332

In [65]: a.std()                # standard deviation of a
Out[65]: 0.21117684925477997

In [66]: a.std(axis = 0)        # standard diviation for each column
Out[66]: array([0.15342784, 0.0049172 , 0.21221967])

In [67]: np.std(a)              # equivalent to a.std()
Out[67]: 0.21117684925477997
```

Fig 2.10 (click to enlarge)

Linear Algebra

Linear algebra, like matrix multiplication, decomposition, determinants, and other square matrix math, is an important part of any array library. See Table 2.3.2 which lists all linear algebra methods:

Table 2.3.2: List of Linear Algebra Methods

Function	Description
diag	Return the diagonal elements of a square matrix as 1-D array

Function	Description
dot	Matrix Multiplication
trace	Compute the sum of the diagonal element
det	Compute the matrix determinant
eig	Compute the eigenvalues and eigenvectors of a square matrix
inv	Compute the inverse of a square matrix
pinv	Compute the Moore-Penrose pseudo-inverse of a square matrix
qr	Compute the QR decomposition
svd	Compute the singular value decomposition
solve	Solve the linear system Ax=b, where A is a square matrix
lstsq	Compute the least=square solution to y=xb

Example

Here are some examples:

```
In [68]: A = np.array( [[1,1], [0,1]] )

In [69]: B = np.array( [[2,0], [3,4]] )

In [70]: A*B                      # elementwise product
Out[70]:
array([[2, 0],
       [0, 4]])

In [71]: A.dot(B)                 # matrix product
Out[71]:
array([[5, 4],
       [3, 4]])

In [72]: np.dot(A, B)             # matrix product
Out[72]:
array([[5, 4],
       [3, 4]])

In [73]: A.T                      # Transpose a matrix
Out[73]:
array([[1, 0],
       [1, 1]])
```

Fig 2.11 (click to enlarge)

Lesson 2.4: Universal Functions: Fast Element-Wise Array Functions (5 of 6)

Lesson 2.4: Universal Functions: Fast Element-Wise Array

Functions

A universal function is a function that performs elementwise operation on data in ndarrays. See Table 2.4.1 for a list of the universal functions:

Table 2.4.1: Universal Functions

Function	Description
abs	Compute the absolute value element-wise
sqrt	Compute the square root of each element
square	Compute the square of each element
exp	Compute the exponential e^x of each element
log, log10, log2, log1p	Natural logarithm (base e), log base 10, log base 2 and log(1 +x), respectively
sign	Compute the sign of each element
ceil	Compute the ceiling of each element
floor	Compute the floor of each element
rint	Round elements to the nearest integer, preserving the dtype
modf	Return factional and integral part of array as separate array
isnan	Return boolean array indicating whether each value is NAN
isfinite, isinf	Return boolean array indicating each element is finite or infinite
cos, cosh, sin, sinh, tan, tanh	Regular and hyperbolic trgonometric functions
arccos, arccosh, arcsin, arcsinh, arctan, arctanh	Inverse trigonometric functions
logical_not	Compute truth value of not

Examples

Click the dropdown to view examples of functions from Table 2.4.1

```
In [74]: A = np.arange(12).reshape((3,4))

In [75]: np.sqrt(A)              # compute square root of each element
Out[75]:
array([[0.        , 1.        , 1.41421356, 1.73205081],
       [2.        , 2.23606798, 2.44948974, 2.64575131],
       [2.82842712, 3.        , 3.16227766, 3.31662479]])

In [76]: B = np.exp(A)           # compute the exponential e^x for each element

In [77]: np.modf(B)              # compute factional and integral part of array
Out[77]:
(array([[0.        , 0.71828183, 0.3890561 , 0.08553692],
        [0.59815003, 0.4131591 , 0.42879349, 0.63315843],
        [0.95798704, 0.08392758, 0.46579481, 0.1417152 ]]),
 array([[1.0000e+00, 2.0000e+00, 7.0000e+00, 2.0000e+01],
        [5.4000e+01, 1.4800e+02, 4.0300e+02, 1.0960e+03],
        [2.9800e+03, 8.1030e+03, 2.2026e+04, 5.9874e+04]]))

In [78]: np.floor(B)             # compute floor of each element
Out[78]:
array([[1.0000e+00, 2.0000e+00, 7.0000e+00, 2.0000e+01],
       [5.4000e+01, 1.4800e+02, 4.0300e+02, 1.0960e+03],
       [2.9800e+03, 8.1030e+03, 2.2026e+04, 5.9874e+04]])
```

Fig 2.12 (click to enlarge)

Lesson 2 References

- “Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython”, Chapter 4, Author Wes McKinney, Publisher "O'Reilly Media, Inc.", 2012
- https://docs.scipy.org/doc/numpy/user/quickstart.html (https://docs.scipy.org/doc/numpy/user/quickstart.html)