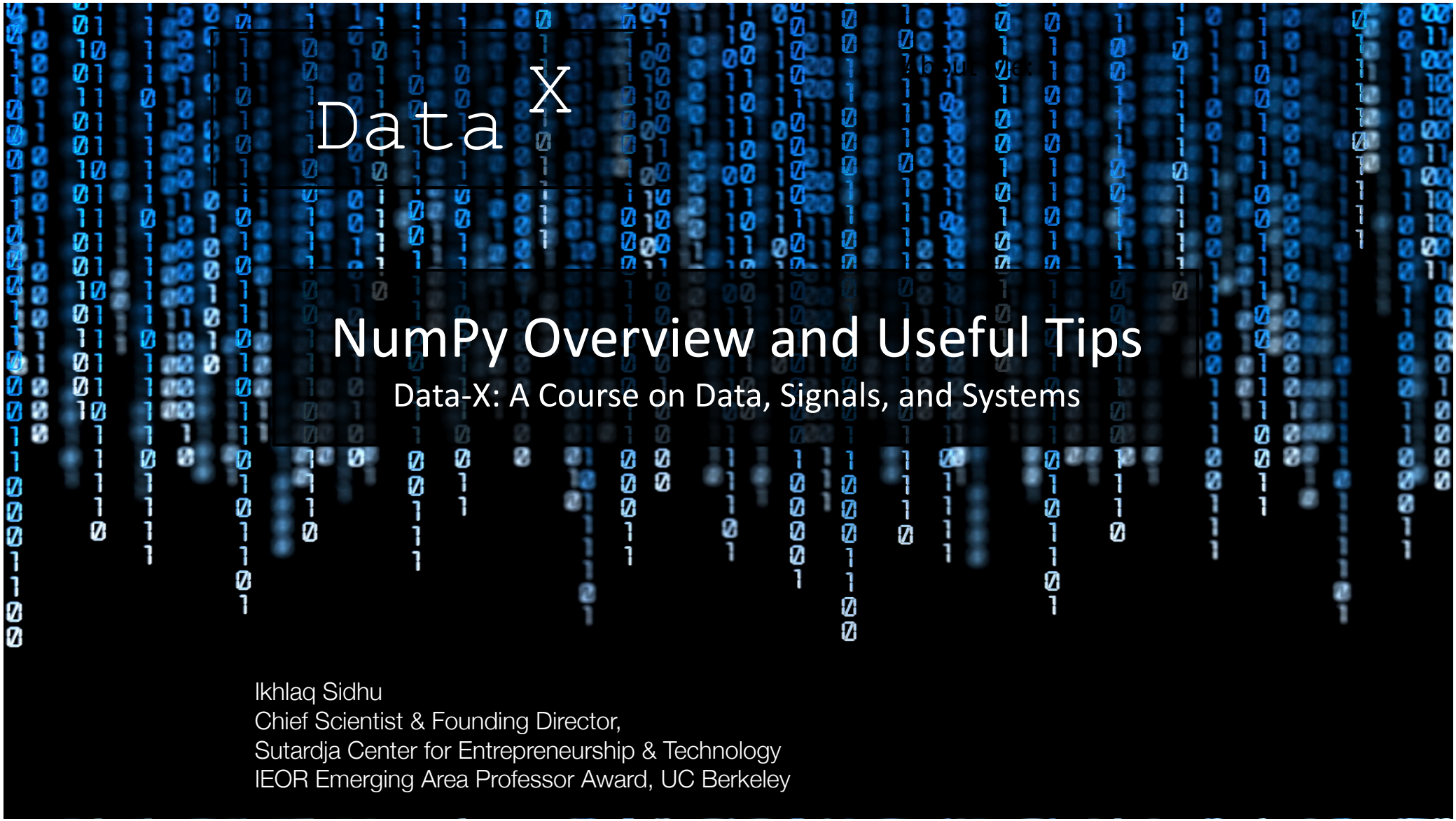


NumPy Overview and Useful Tips

Data-X: A Course on Data, Signals, and Systems

Ikhlaq Sidhu
Chief Scientist & Founding Director,
Sutardja Center for Entrepreneurship & Technology
IEOR Emerging Area Professor Award, UC Berkeley

- NumPy Overview
  - Numpy Notebook, Slides
  - See Ref CS02: NumPy Getting Started v1-12

Arrays using NumPy

- Helpful tips beyond the Numpy Notebook
- Common mistakes

For more information, please review at:
https://docs.scipy.org/doc/numpy-dev/user/quickstart.html
See section 3 to understand strings and list

Create a NumPy Array

Creating arrays in NumPy, however, is different from creating the array.array library in Python. Here is how you do it:

```
>>> a = np.array(1,2,3,4)     # WRONG
>>> a = np.array([1,2,3,4])   # RIGHT
```

Multidimensional Arrays

NumPy arrays can also be multidimensional

- `a = np.array([ [1, 2, 3], [4, 5, 6] ])`
- `b = np.array([ [7, 8, 9], [10, 11, 12] ])`
  `a` and `b` are two 2D arrays (NumPy matrices)
- `a.shape` Returns the array's dimensions, here: `(2, 3)`

Basic math work on multidimensional arrays

- `a+b, a-b, a*b, a/b` performs *elementwise* addition, subtraction, multiplication, and division
- `np.dot(np.transpose(a),b)` performs matrix multiplication of a^T and b

Use linespace for evenly spaced points in a range

Suppose you want an array with 9 points evenly spaced from 0 to 2?

Use linespace for evenly spaced points in a range

Suppose you want an array with 9 points evenly spaced from 0 to 2?

```
                                                                    >>>
>>> from numpy import pi
>>> np.linspace( 0, 2, 9 )                  # 9 numbers from 0 to 2
array([ 0.  ,  0.25,  0.5 ,  0.75,  1.  ,  1.25,  1.5 ,  1.75,  2.  ])
>>> x = np.linspace( 0, 2*pi, 100 )         # useful to evaluate function at lot
s of points
>>> f = np.sin(x)
```

Comparisons

```
a = np.array( [20,30,40,50] )
```

```
>>> 10*np.sin(a)
array([ 9.12945251, -9.88031624,  7.4511316 , -2.62374854])
>>> a<35
array([ True, True, False, False], dtype=bool)
```

Also known as a mask:
m = a<35
a(m)

If You need Random Numbers, no problem.
Look up np.random.random

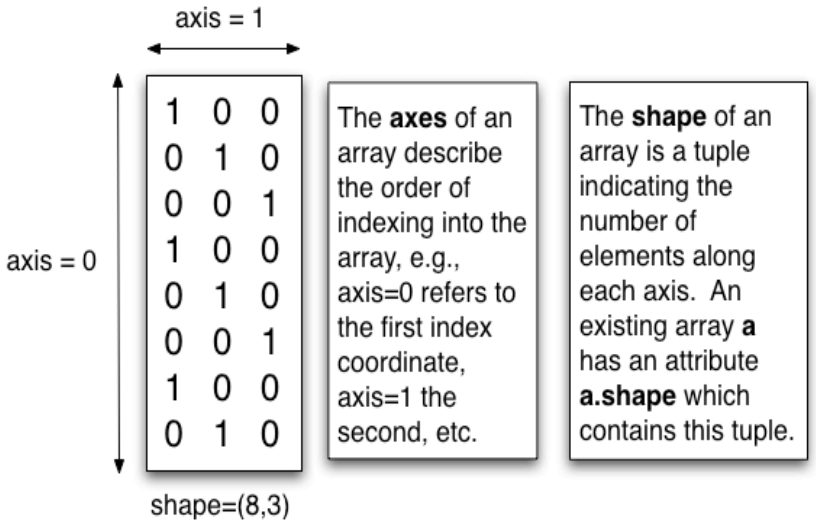
```
>>> a = np.random.random((2,3))
>>> a
array([[ 0.18626021,  0.34556073,  0.39676747],
       [ 0.53881673,  0.41919451,  0.6852195 ]])
>>> a.sum()
2.5718191614547998
>>> a.min()
0.18626021134709
>>> a.max()
0.6852195003967595
```

Also see

- np.zeros and np.ones
- https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.random.html

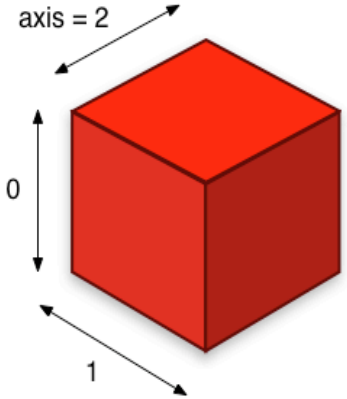
Anatomy of an Array

In NumPy, dimensions are called axes and the number of axes is called a rank.



The **axes** of an array describe the order of indexing into the array, e.g., axis=0 refers to the first index coordinate, axis=1 the second, etc.

The **shape** of an array is a tuple indicating the number of elements along each axis. An existing array **a** has an attribute **a.shape** which contains this tuple.



- all elements must be of the same dtype (datatype)

- the default dtype is float

- arrays constructed from list of mixed dtype will be upcast to the "greatest" common type

Using the axis parameter
with methods like sum, min, and others

```
>>> b = np.arange(12).reshape(3,4)
>>> b
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

Using the axis parameter with methods like sum, min, and others

```
>>> b = np.arange(12).reshape(3,4)
>>> b
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>>
>>> b.sum(axis=0)                          # sum of each column
array([12, 15, 18, 21])
>>>
```

Using the axis parameter with methods like sum, min, and others

```
>>> b = np.arange(12).reshape(3,4)
>>> b
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>>
>>> b.sum(axis=0)                          # sum of each column
array([12, 15, 18, 21])
>>>
>>> b.min(axis=1)                          # min of each row
array([0, 4, 8])
>>>
```

Using the axis parameter with methods like sum, min, and others

```
>>> b = np.arange(12).reshape(3,4)
>>> b
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>>
>>> b.sum(axis=0)                          # sum of each column
array([12, 15, 18, 21])
>>>
>>> b.min(axis=1)                          # min of each row
array([0, 4, 8])
>>>
>>> b.cumsum(axis=1)                       # cumulative sum along each row
array([[ 0,  1,  3,  6],
       [ 4,  9, 15, 22],
       [ 8, 17, 27, 38]])
```

Slicing a NumPy Array, 1-Dimension

```
>>> x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> x[1:7:2]


array([1, 3, 5])
```

start:stop:step
i:j:k

Slicing a NumPy Array, 1-Dimension

```
>>> x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> x[1:7:2]


array([1, 3, 5])
```

start:stop:step
i:j:k

```
>>> x[-2:10]
array([8, 9])
>>> x[-3:3:-1]
array([7, 6, 5, 4])
```

Negative *i* and *j* are interpreted as *n* - *i* and *n* - *j* where *n* is the number of elements in the corresponding dimension. Negative *k* makes stepping go towards smaller indices.

Slicing a NumPy Matrix Works the Same Way
You get a new "view" of a section of the of the original matrix

```
data = np.random.random((1000,4))
print data

[[ 0.81946001 0.24377412 0.95235583 0.58232243]
 [ 0.58229757 0.12284915 0.83731775 0.52798616]
 [ 0.6921377 0.08912797 0.39800043 0.66288952]
 ...,
 [ 0.83346456 0.20024193 0.86708519 0.09500233]
 [ 0.27112654 0.72018064 0.17056836 0.26202908]
 [ 0.22147895 0.27752412 0.52697043 0.36483595]]
```

data[1:6:1, 0:3:1]

```
array([
[ 0.58229757, 0.12284915, 0.83731775],
[ 0.6921377 , 0.08912797, 0.39800043],
[ 0.84952767, 0.1318117 , 0.61345601],
 [ 0.3038797 , 0.81956695, 0.9835471 ],
[ 0.15578211, 0.99188135, 0.63214512]])
```

Rows 2 to 6, Columns 1 to 3
Same as data[1:6,0:3]

data[1:6,0:3] = 0, will assign those as 0

More notation:
b[0:5,1] gives you rows 0:5, and column 2
b[:,1] give you the same thing if b has 5 rows

Delete for Slicing a NumPy array (matrix)

The simplest way to delete rows and columns from arrays is the `numpy.delete` method.

Suppose I have the following array `x`:

```
x = array([[1,2,3],
           [4,5,6],
           [7,8,9]])
```

To delete the first row, do this:

```
x = numpy.delete(x, (0), axis=0)
```

To delete the third column, do this:

```
x = numpy.delete(x,(2), axis=1)
```

So you could find the indices of the rows which have a 0 in them, put them in a list or a tuple and pass this as the second argument of the function.

share improve this answer

edited Feb 15 '15 at 19:48
MERose
1,163 ● 2 ● 13 ● 32

answered Jul 26 '12 at 5:48
Jaidev Deshpande
850 ● 7 ● 16

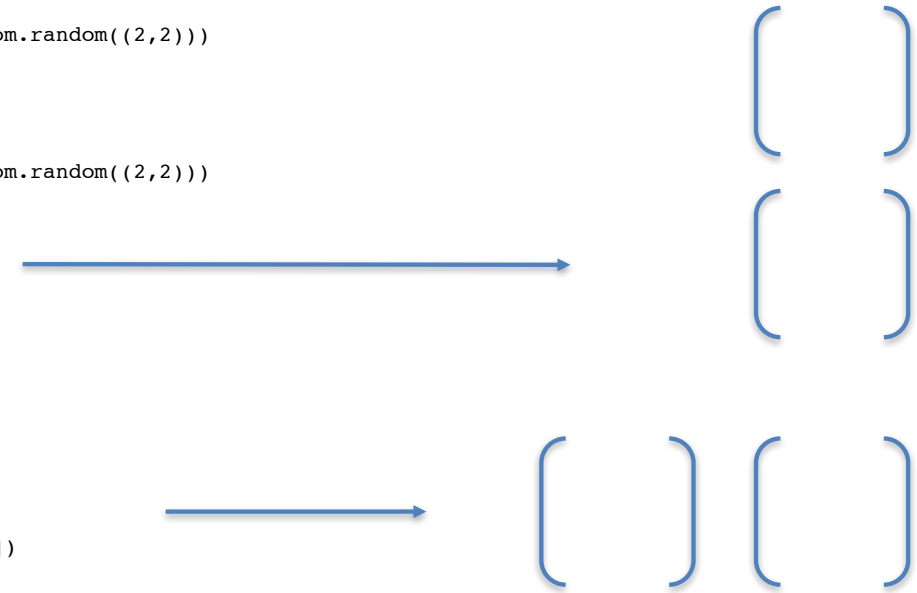
But be careful, it will really be gone.

From stackoverflow

Stacking Matrices: See hstack and vstack

Several arrays can be stacked together along different axes:

```
>>> a = np.floor(10*np.random.random((2,2)))
>>> a
array([[ 8.,  8.],
       [ 0.,  0.]])
>>> b = np.floor(10*np.random.random((2,2)))
>>> b
array([[ 1.,  8.],
       [ 0.,  4.]])
>>> np.vstack((a,b))
array([[ 8.,  8.],
       [ 0.,  0.],
       [ 1.,  8.],
       [ 0.,  4.]])
>>> np.hstack((a,b))
array([[ 8.,  8.,  1.,  8.],
       [ 0.,  0.,  0.,  4.]])
```



Also see:

- numpy.column_stack: its like hstack, but for 1 column at a time
- numpy.concatenate: join a sequence of arrays along an existing axis

You can also split matrices: hsplit and vsplit

Using hsplit (../reference/generated/numpy.hsplit.html#numpy.hsplit), you can split an array along its horizontal axis, either by specifying the number of equally shaped arrays to return, or by specifying the columns after which the division should occur:

```
>>> a = np.floor(10*np.random.random((2,12)))
>>> a
array([[ 9.,  5.,  6.,  3.,  6.,  8.,  0.,  7.,  9.,  7.,  2.,  7.],
       [ 1.,  4.,  9.,  2.,  2.,  1.,  0.,  6.,  2.,  2.,  4.,  0.]])
>>> np.hsplit(a,3)   # Split a into 3
[array([[ 9.,  5.,  6.,  3.],
       [ 1.,  4.,  9.,  2.]]), array([[ 6.,  8.,  0.,  7.],
       [ 2.,  1.,  0.,  6.]]), array([[ 9.,  7.,  2.,  7.],
       [ 2.,  2.,  4.,  0.]])]
>>> np.hsplit(a,(3,4))   # Split a after the third and the fourth column
[array([[ 9.,  5.,  6.],
       [ 1.,  4.,  9.]]), array([[ 3.],
       [ 2.]]), array([[ 6.,  8.,  0.,  7.,  9.,  7.,  2.,  7.],
       [ 2.,  1.,  0.,  6.,  2.,  2.,  4.,  0.]])]
```

vsplit (../reference/generated/numpy.vsplit.html#numpy.vsplit) splits along the vertical axis, and array_split (../reference/generated/numpy.array_split.html#numpy.array_split) allows one to specify along which axis to split.

Flatten (use ravel) and the reshape array:

```
>>> a = np.floor(10*np.random.random((3,4)))
>>> a
array([[ 2.,  8.,  0.,  6.],
       [ 4.,  5.,  1.,  1.],
       [ 8.,  9.,  3.,  6.]])
>>> a.shape
(3, 4)
```

The shape of an array can be changed with various commands:

```
>>> a.ravel() # flatten the array
array([ 2.,  8.,  0.,  6.,  4.,  5.,  1.,  1.,  8.,  9.,  3.,  6.])
>>> a.shape = (6, 2)
>>> a.T
array([[ 2.,  0.,  4.,  1.,  8.,  3.],
       [ 8.,  6.,  5.,  1.,  9.,  6.]])
```

A Note on Copies

Assignment:

No Copy at All

Simple assignments make no copy of array objects or of their data.

```
>>> a = np.arange(12)
>>> b = a            # no new object is created
>>> b is a           # a and b are two names for the same ndarray object
True
>>> b.shape = 3,4    # changes the shape of a
>>> a.shape
(3, 4)
```

A Note on Copies

No Copy at All

Simple assignments make no copy of array objects or of their data.

Assignment:

```
>>> a = np.arange(12)
>>> b = a            # no new object is created
>>> b is a           # a and b are two names for the same ndarray object
True
>>> b.shape = 3,4    # changes the shape of a
>>> a.shape
(3, 4)
```

Shallow Copy:

A Slice is a View of the data. Changes format, but the data is still the same

```
>>> s = a[ : , 1:3]     # spaces added for clarity; could also be written "s = a[:,1:3]"
>>> s[:] = 10           # s[:] is a view of s. Note the difference between s=10 and s[:]=10
>>> a
array([[   0,   10,   10,    3],
       [1234,   10,   10,    7],
       [   8,   10,   10,   11]])
>>>
```

Deep Copy

A Note on Copies

Assignment:
No Copy at all
b = a

Shallow Copy:
A Slice is a View of the data.
Changes format, but the data is still the same

Deep Copy:

Deep Copy

The `copy` method makes a complete copy of the array and its data.

```
>>> d = a.copy()                          # a new array object with new data is
 created
>>> d is a
False
>>> d.base is a                           # d doesn't share anything with a
False
>>> d[0,0] = 9999
>>> a
array([[   0,   10,   10,    3],
       [1234,   10,   10,    7],
       [   8,   10,   10,   11]])
```

End of Section