Python for Scientific Computing
Lecture 5: Numpy, Scipy & Matplotlib

Patrick Pisciuneri

Center for Simulation & Modeling
University of Pittsburgh

September 25, 2013
Quick Highlights:

▶ Fundamental package for scientific computing with Python.
▶ High-level number objects for ints, floats, etc.
▶ Provides a powerful N-dimensional array object.
  ▶ Usage will be familiar to Matlab users.

Useful links to get started with Numpy:

▶ http://www.numpy.org/
▶ http://scipy-lectures.github.io/intro/numpy/index.html
Numpy

Import convention:

```python
>>> import numpy as np
```

What are working with:

```python
>>> np?
```

Straight from the docstring:

1. An array object of arbitrary homogeneous items
2. Fast mathematical operations over arrays
3. Linear Algebra, Fourier Transforms, Random Number Generation
Numpy: Arrays

Creating an array:

```python
>>> np.array( [0, 1, 2, 3, 4, 5] )
array([0, 1, 2, 3, 4, 5])
```

Creating an array of evenly spaced values:

```python
>>> np.arange( 6 )
array([0, 1, 2, 3, 4, 5])
```

Specifying start, stop, interval and data type:

```python
>>> np.arange( 2, 20, 2, dtype=float )
array([ 2., 4., 6., 8., 10., 12., 14., 16., 18.])
```
Numpy: Efficiency

List:

```python
>>> lst = range(100000)
>>> %timeit [ i**2 + np.sin(i) for i in lst ]
1 loops, best of 3: 602 ms per loop
```

Numpy array (no looping necessary):

```python
>>> arr = np.arange(100000)
>>> %timeit arr**,2 + np.sin(arr)
100 loops, best of 3: 2.29 ms per loop
```

For this example, that is approx. 260 times faster!
Numpy: Arrays

Creating a 2D array:

```python
>>> a = np.array( [ [0, 1, 2], [3, 4, 5] ] )
```

```python
>>> a
array([[0, 1, 2],
       [3, 4, 5]])
```

Attributes:

```python
>>> a.ndim  # returns number of dimensions
2
>>> a.shape  # returns (rows, columns)
(2, 3)
>>> a.size  # total number of elements
6
>>> len(a)  # returns size of first dimension only
2
```
Numpy: Arrays - reshaping

Begin with a 1D array:

```
>>> b = np.arange(6)
```

Functional approach:

```
>>> np.reshape(b, (2,3))
array([[0, 1, 2],
       [3, 4, 5]])
```

Object-oriented approach:

```
>>> b.reshape(2,3)
array([[0, 1, 2],
       [3, 4, 5]])
```

All in one step:

```
>>> c = np.arange(6).reshape((2,3))
>>> c
array([[0, 1, 2],
       [3, 4, 5]])
```
Numpy: Arrays - some special examples

```python
>>> a = np.zeros((2,2))
>>> a
array([[0., 0.],
       [0., 0.]])
>>> b = np.ones((2,2))
>>> b
array([[1., 1.],
       [1., 1.]])
>>> c = np.eye(2)
>>> c
array([[1., 0.],
       [0., 1.]])
>>> d = np.diag([1,2,3])
>>> d
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 3]])
```
What generators are available?

```python
>>> np.random?

Normal RNG:

```python
>>> a = np.random.randn(100000)
>>> a.mean()
-0.0058503425037383424
>>> a.std()
1.0011225635438217
```

Uniform RNG:

```python
>>> b = np.random.rand(100000)
>>> b.mean()
0.50032310554491544
>>> b.std()
0.28934098515418233
```
Numpy: Arrays - slicing

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

>>> c = a[1], a[6], a[-2]

>>> c
(1, 6, 8)

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

>>> b = np.diag( c )

>>> b
array([[1, 0, 0],
        [0, 6, 0],
        [0, 0, 8]])

>>> b[1, 1]
6
```
Slicing: Defaults start(0), end(last), step(1)

```python
>>> a = np.arange(10)
>>> a[:6]  # Non-default: end
array([0, 1, 2, 3, 4, 5])
```

**Note:** When the end index is specified, it is not included in the slice!

```python
>>> a[4:7]  # Non-default: start and end
array([4, 5, 6])
```
```python
>>> a[::3]  # Non-default: step
array([0, 3, 6, 9])
```
```python
>>> a[5:]  # Non-default: start
array([5, 6, 7, 8, 9])
```
Numpy: Arrays - views vs. copies

```python
>>> a = np.arange( 10 )
>>> b = a.reshape( 2, 5 )
>>> c = a.reshape( 2, 5 ).copy()

>>> a[0] = 99
>>> b
array([[99, 1, 2, 3, 4],
       [ 5, 6, 7, 8, 9]])

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

When views? - Examine the help for a particular function, or examine:

```python
>>> np.may_share_memory()
```
Numpy: Exercises

For some practice with indexing and slicing, array creation, and tiling for array creation please complete the three exercises available from the Scipy-Lecture-Notes:

http://scipy-lectures.github.io/intro/numpy/array_object.html#indexing-and-slicing
Quick Highlights:

- High-level scientific computing.
- Various toolboxes dedicated to common issues in scientific computing.
  - Similar to the toolboxes and functionality offered by Matlab.
- Offers routines that are highly optimized and tested.

Useful links to get started with Scipy:

- http://www.scipy.org/
## Scipy

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

Example:

\[ x - y = 2 \]
\[-.2x - y = -4 \]

Solution:

\[ x = 5 \]
\[ y = 3 \]

Matrix / Vector form:

\[ \begin{bmatrix} 1 & -1 \\ -.2 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2 \\ -4 \end{bmatrix} \]
```python
>>> import numpy as np
>>> from scipy import linalg
>>> a = np.array( [ [1, -1], [-2, -1] ] )
>>> a
array([[ 1., -1.],
       [-2, -1.]])
>>> linalg.det( a )
-1.2
>>> ainv = linalg.inv( a )
>>> ainv
array([[ 0.83333333, -0.83333333],
       [-0.16666667, -0.83333333]])
>>> b = np.array( [ [2], [-4] ] )
>>> x = np.dot( ainv, b )
>>> x
array([[ 5.],
       [ 3.]])
```
Or we can just use a built-in linear system solver:

```python
>>> a = np.array( [[1, -1], [-2, -1]] )
>>> b = np.array( [[2], [-4]] )
>>> x = linalg.solve( a, b )
>>> x
array([[ 5.],
       [ 3.]]
```

Examine the help!

```python
>>> linalg?
```

Examples using other toolboxes:
Matplotlib

Quick Highlights:

- Python 2D plotting library which produces publication quality figures.
- Use ipython with pylab to have an interactive environment similar to Matlab or Mathematica.
- Offers functionality for full control of line styles, font properties, axes properties, etc.

Useful links to get started with Scipy:

- http://matplotlib.org/
Matplotlib

Import convention:

```python
>>> import numpy as np
>>> import matplotlib.pyplot as plt
```

Interactive mode (from the command line):

```python
# ipython --pylab
```

Interactive mode (within IPython GUI):

```python
>>> %pylab
```

**WARNING:** Starting IPython in interactive mode executes the following statements (among others):

```python
>>> from pylab import *
>>> from numpy import *
```

so your namespace will be polluted. For interactive sessions this may be tolerable, but in general following the import convention above and placing plot routines in scripts is better practice.
import numpy as np
import matplotlib.pyplot as plt

x = np.random.randn(100000)

plt.hist(x, 100)
plt.show()
Matplotlib: Figure properties

```python
import numpy as np
import matplotlib.pyplot as plt

x = np.random.randn(100000)
plt.figure(figsize=(8,8))
plt.hist(x, 100, color='yellow')
plt.xlabel('x', fontsize=20, fontweight='bold')
plt.ylabel('y', fontsize=20, fontweight='bold')
plt.title('Normal Distribution', fontsize=24, fontweight='bold')
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.box('on')
plt.show()
```

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(3, 7, 25)
y1 = x - 2
y2 = -0.2*x + 4
plt.plot(x, y1, label='y1', linewidth=4, marker='o', markersize=10)
plt.plot(x, y2, label='y2', linewidth=4, marker='d', markersize=10)
plt.xlabel('x', fontsize=20, fontweight='bold')
plt.ylabel('y', fontsize=20, fontweight='bold')
plt.xticks([3, 4, 5, 6, 7], fontsize=20)
plt.yticks([1, 2, 3, 4, 5], fontsize=20)
plt.legend(loc='upper left')
plt.box('on')
plt.grid('on')
plt.show()
```