The SciPy Stack

Jay Summet

May 1, 2014
Outline

• Numpy - Arrays, Linear Algebra, Vector Ops
• MatPlotLib - Data Plotting
• SciPy - Optimization, Scientific functions
What is Numpy?

- 3rd party module that extends the Python language
- Matrix and linear algebra operations

General support for N-Dimensional Arrays
  - Heterogeneous elements
  - Direct mapping to C/C++/Fortran memory buffers
  - Matrix operations
  - Vectorized Algorithms

Bindings to C and Fortran matrix math libraries
  - Requires compiled support libraries
  - Installers are platform specific
Importing Numpy

- Must install it via the SciPy stack or individually
- Most people import the numpy library “as” np
- Arrays can be created from lists or nested lists

```python
>>> import numpy as np
>>> x = np.array([1, 5, 10])
>>> x
array([1, 5, 10])

>>> y = np.array([[1, 2, 3], [4, 5, 6]])
>>> y
array([[1, 2, 3],
       [4, 5, 6]])
```
Data Types

• Numpy stores all data in an array using a single data type
• Direct mapping to memory / space required
• The Python data types have default mappings
• You will most commonly use:
  – bool - boolean, stored as a byte
  – int_ (Same as C long, either int32 or int64, architecture dependent)
  – float_ ( float64)
  – complex_ (complex128)

• Will convert python int/float/bool/complex automatically
Data Types

- Numpy supports many different native data types
- Will convert python int/float/bool/complex automatically
- The .dtype attribute of an array tells the actual data type
- Can convert to another data type using .astype()

```python
>>> x = np.array([1, 5, 10])
>>> x
array([1, 5, 10])
>>> x.dtype
dtype('int32')
>>> y = x.astype(np.float64)
>>> y
array([ 1.,  5.,  10.])
>>> y.dtype
dtype('float64')
```
Numpy Array Creation

• Convert python “array-like” data structures
• Using built in array creation methods:
  – `np.zeros( shape )`
  – `np.ones( shape )`
  – `np.arange(Start, Stop, step)` - Like python range
  – `np.linspace( Start, Stop, number)`
• Most accept an optional `dtype` parameter

```python
>>> x = np.zeros( (2,2), dtype=np.int_
>>> x
array([[0, 0],
       [0, 0]])

>>> y = np.linspace( 1, 4, 6 )
>>> y
array([1., 1.6, 2.2, 2.8, 3.4, 4.])
```
Array = Block of Memory

- All data in the array is stored in a single block of memory
- The .shape attribute stores the size of each dimension
- You can change the shape on the fly!
  - must retain the exact number of elements
  - A 1 x 10 array and a 2 x 5 array both have 10 data items

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

- **Works like indexing in python**
  - Zero based
  - negative indexes count from the back
- **BUT, You CAN index all dimensions at once if you want!**

```python
>>> x = np.arange(10)
>>> x[-1]
9
>>> x.shape = (2,5)
>>> x[1][3]
8
>>> x[1,3]
8
```
Array Slicing

- **Works like indexing in python**
  - Provide a Start:Stop:Stride
  - Can do this for multiple dimensions at once

```python
>>> x = np.arange(25).reshape(5,5)
>>> x
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])

>>> x[0:5:2, 2:4]
array([[2, 3],
       [12, 13],
       [22, 23]])
```
Array Slicing - Aliasing!

• Slices provide VIEWS
  – Like an alias of a list

• See array indexing if you want to copy a subset of data!

>>> y = x[0:5:2, 2:4]
>>> y[0,0] = 999
>>> y
array([[999, 3],
       [12, 13],
       [22, 23]])

>>> x
array([[ 0,  1,  999,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])
Advanced Indexing

- You can specify a set of indices with an array to select a copy of the data.

```python
>>> x
array([ 0,  5, 10, 15, 20, 25])

>>> y = x[np.array([1,3,5])]
>>> y
array([ 5, 15, 25])

>>> y[0] = 99
>>> y
array([99, 15, 25])

>>> x
array([ 0,  5, 10, 15, 20, 25])
```
Advanced Indexing

• BUT, if you assign to an array indexing operation, it modifies the original...

```python
>>> x
array([ 0,  5, 10, 15, 20, 25])

>>> y = x[ np.array( [1,3,5] ) ]
>>> y
array([ 5, 15, 25])

>>> x[ np.array( [1,3,5] ) ] += 1
>>> x
array([ 0,  6, 10, 16, 20, 26])
```
Array Broadcasting

- Element-by-element operations on two arrays typically require the arrays to have the same shape

- A scaler is treated as a 1x1 array that gets “broadcast” over all elements

```python
>>> y = np.arange(1,5)
>>> y
array([1, 2, 3, 4])
>>> y * 2
array([2, 4, 6, 8], dtype=int32)
>>> y * 2.
array([2., 4., 6., 8.])
```
Array Broadcasting

- Arrays that are “short” in a dimension can also be broadcast!
  - The Rule: “In order to broadcast, the size of the trailing axes for both arrays in an operation must either be the same size or one of them must be one.”

```python
c = a * b
```

```python
c = np.array( [[0, 1, 2], [3, 4, 5]] )
>>> b = np.array([4,5,6])
>>> c = a * b
>>> c
array([[  0,   5,  12],
       [ 12,  20,  30]], dtype=int32)
```
Vector Algorithms:

- Calculating $y = \sin(x)$, for 100 elements between -4 and 4

```python
>>> x = np.linspace(-4,4,100)
>>> y = np.sin(x)
```
What is Matplotlib?

• Python library that supports plotting data

• Many features were copied from MATLAB

• Easily produce Graphs/Plots of
  • Lines
  • Histograms
  • Bargraphs
  • Scatterplots
  • 3D surfaces
Plotting sin data

• Plotting 2D data (x vs y) requires two lines of code
  – Three if you count the import....

```python
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(-4,4,100)
y = np.sin(x)
temp = plt.plot(x,y)
plt.show()
```
Filled Plot

```python
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(-np.pi, np.pi, 100)
y = np.sin(x)
temp = plt.fill(x, y, color="green")
plt.grid(True)
plt.show()
```
Other plots

• Pie charts

```python
import matplotlib.pyplot as plt

labels = ['Red', 'Green', 'Blue', 'Yellow']
colors = ['red', 'lightgreen', 'lightskyblue', 'gold']
values = [3, 4, 10, 1]
explode = (0.2, 0, 0, 0)

plt.pie(values, explode=explode, colors=colors,
        labels=labels, shadow=True, autopct='%.1f%%')

plt.show()
```
3D Plots require 3D data

• Numpy meshgrid function allows you to create arrays with all possible X or Y coordinates

```python
>>> x = np.arange(0,5,1)
>>> y = np.arange(0,4,1)
>>> x
array([0, 1, 2, 3, 4])
>>> y
array([0, 1, 2, 3])
>>> NX,NY = np.meshgrid(x,y)
>>> NX
array([[0, 1, 2, 3, 4],
       [0, 1, 2, 3, 4],
       [0, 1, 2, 3, 4],
       [0, 1, 2, 3, 4]])
>>> NY
array([[0, 0, 0, 0, 0],
       [1, 1, 1, 1, 1],
       [2, 2, 2, 2, 2],
       [3, 3, 3, 3, 3]])
```
3D Plots require 3D data

- These arrays can be used to calculate a 3rd (Z) axis based upon a formula, such as euclidean distance from origin.
  - Note the distance at point (4,3) is 5 (3x4x5 triangle)

```python
>>> D = np.sqrt( NX**2 + NY**2 )
>>> D
array([[ 0.        ,  1.        ,  2.        ,  3.        ,  4.        ],
       [ 1.        ,  1.4142,  2.2360,  3.1622,  4.1231],
       [ 2.        ,  2.2360,  2.8284,  3.6055,  4.4721],
       [ 3.        ,  3.1622,  3.6055,  4.2426,  5.        ]])
```
3D Plots require 3D data

• And you could calculate the sin of the distance...

```python
>>> Z = np.sin(D)
>>> Z
```

```
array([[ 0.        ,  0.84147,  0.90929,  0.14112, -0.7568     ],
       [ 0.84147,  0.98776,  0.78674, -0.02068, -0.83133 ],
       [ 0.90929,  0.78674,  0.30807, -0.44749, -0.97127 ],
       [ 0.14112, -0.02068, -0.44749, -0.89168, -0.95892 ]])
```
3D surface Plot

```python
>>> import matplotlib.pyplot as plt
>>> from mpl_toolkits.mplot3d import Axes3D
>>> fig = plt.figure()
>>> ax = fig.gca(projection='3d')  #Requires Axes3D import!
>>> surf = ax.plot_surface(NX,NY, Z, rstride=1, cstride=1, linewidth=0)
>>> plt.show()
```
import numpy as np

X = np.arange(-5, 5, 0.1)
Y = np.arange(-5, 5, 0.1)
NX,NY = np.meshgrid(X, Y)
D = np.sqrt(NX**2 + NY**2)
Z = np.sin(D)

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.gca(projection='3d')
surf = ax.plot_surface(NX,NY, Z, rstride=1, cstride=1, linewidth=0)
plt.show()
Adding more bling...

- Add a colormap and colorbar (key)
- Customizing the Z axis
from matplotlib.ticker import LinearLocator, FormatStrFormatter
from matplotlib import cm

... 
as = fig.gca(projection='3d')

surf = ax.plot_surface(NX, NY, Z, rstride=1, cstride=1, linewidth=0, cmap=cm.coolwarm)

# Colorbar requires the surface to have a colormap.
fig.colorbar(surf, shrink=0.5, aspect=5)

# Customize Z-axis
ax.set_zlim(-1.01, 1.01)
ax.zaxis.set_major_locator(LinearLocator(10))
ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))
What is SciPy?

- Python library that builds upon Numpy arrays
- Provides higher level math support for
  - Optimization
  - Linear Algebra
  - Interpolation
  - FFT
  - Ordinary Differential Equation Solvers
SciPy

- Function minimization
- Classic example function: Rosenbrock

\[ f(x, y) = (1 - x)^2 + 100(y - x^2)^2. \]

Source code in handout: rosenbrock_demo.py
Function Minimization

- Define the function to accept a sequence (of parameters)

```python
def rosenbrock(parms):
    x, y = parms
    return (1-x)**2 + 100 * (y-x**2)**2
```

Source code in handout: rosenbrock_minimum.py
Function Minimization

- Import and call `fmin` from `scipy.optimize`
- Provide a starting point, and GO!

```python
import numpy as np
from scipy.optimize import fmin

x0 = np.array([4, -3])  # Try starting at (4, -3)
res = fmin(rosenbrock, x0, xtol=1e-8)

print(res)
```

Source code in handout: rosenbrock_minimum.py
Function Minimization

Optimization terminated successfully.
Current function value: 0.000000
Iterations: 90
Function evaluations: 174

[ 1.  1.]

Source code in handout: rosenbrock_minimum.py
Questions?