CME 193: Introduction to Scientific Python
Lecture 4: NumPy and SciPy

Dan Frank

Institute for Computational and Mathematical Engineering (ICME)

January 21, 2014

NumPy

SciPy

What is NumPy?

Wikipedia: NumPy is an extension to the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large library of high-level mathematical functions to operate on these arrays.

- At the core of the NumPy package, is the *ndarray* object which encapsulates n-dimensional arrays of homogeneous data.
- Many operations performed using *ndarray* objects execute in compiled code for performance
- The standard mathematical and scientific packages in Python use NumPy arrays

Array Creation

Several ways to create arrays...

```
import numpy as np

#  lists
arr = np.array([[1, 2, 3], [4, 5, 6]])
#array([[1, 2, 3],
#       [4, 5, 6]])

#  sequences
np.arange(0, 10, 0.1)
np.linspace(0, 2 * np.pi, 100)

#  zeros & ones
np.zeros((5, 5))
np.ones((5, 5))
np.diag(np.arange(5))

#  random
np.random.random(size=(3, 4))
np.random.normal(loc=10., scale=3., size=(3, 4, 5))
```

Array IO

```
# create an array, write to file, read from file
arr = np.array([[1, 2, 3], [4, 5, 6]])

# save to a text file
# creates a space delimited file by default
np.savetxt(fname='array_out.txt', X=arr)

# load text file
loaded_arr = np.loadtxt(fname='array_out.txt')

np.all(arr == loaded_arr)  # True
```

Other options control data types, delimiters, comments, headers, etc. See documentation, especially "See Also". And for matlab `scipy.io.matlab`

Array Attribtues

Arrays are objects and so have attributes and methods.

```
arr = np.arange(10).reshape((2, 5))

arr.ndim   #  2 number of dimensions
arr.shape  #  (2, 5) shape of the array
arr.size   #  10 number of elements
arr.T      #  transpose
arr.dtype  #  data type of elements in the array
```

And many others. Explore in documentation or with TAB complete in ipython.

Array Slicing and Indexing

Similar to lists but a few new ways to select

```
arr = np.arange(20).reshape((4, 5))

# slicing (like lists for each dimension)
arr[0:4, 3:5]  # all rows and last two columns
arr[:4,  3:5]  # equivalent - can leave off start if 0
arr[:, 3:]     # equivalent - can leave off end if size of axis
arr[slice(None), slice(3, None)]  # equivalent - can use slice()

# integer indices
arr[[1, 2], :]  # rows one and two, all columns
arr[np.array([1, 2]), :]  # equivalent
arr[[1, 2], [1, 2, 3]] # ERROR

# boolean indices
arr[np.array([False, True, True, False]), :]  # equivalent
arr[[False, True, True, False], :]
# NOT equivalent - boolean treated as 1, 0
```

Array Slicing and Indexing

Different indexing can be very powerful, used for extraction and setting of data.

```
arr = np.arange(-100, 50)

arr[arr > 0] # extracts positive part of the array

arr[arr > 0] = 0 # sets positive part of array to 0

# combined!
arr[arr > 0] = np.random.random_integers(low=0,
                                         high=10,
                                         size=len(arr[arr > 0]))
```

Array Axes

Arrays are organized by their axes, the first being row, then column, ...

```
arr = np.random.random((2, 5))

arr.sum() # sum of all elements in array
np.sum(arr) # equivalent
np.sum(arr, axis=0) # column sum
np.sum(arr, axis=1) # row sum

np.apply_along_axis(np.sum, 0, arr) # column sum

arr3d = np.random.random((2, 3, 4))
np.sum(arr3d, axis=2) # returns a (2, 3) array
```

Many functions in NumPy and SciPy are created to operate along axes

Array Operations & ufuncs

Default behavior is elementwise

```
arr1 = np.arange(10).reshape((2, 5))
arr2 = np.random.random((2, 5))

# elementwise for basic and boolean operations
#   +, -, *, /, **, np.log, <, >=, ==
#   arrays are upcast, resulting in float or boolean arrays
arr1 + arr2  # elementwise sum
arr1 == arr2  # elementwise equality

# operations in place
arr1 += arr2

# matrix product
np.dot(arr1, arr2)

# similarly numpy ufunc's operate elementwise
np.sin(arr1)
np.sqrt(arr1)
```

Array Broadcasting & Vectorization

Broadcasting allows us to operate on arrays of different shapes by reusing smaller arrays when possible. This allows us to write more efficient and readable code (with fewer for loops).

```
# multiplication by a scalar
arr = np.random.random((4, 5))
arr * 5  # multiply each element of the array by 5

# scales the first column by 0.
# scales the second column by 1., etc.
arr * np.arange(5)
```

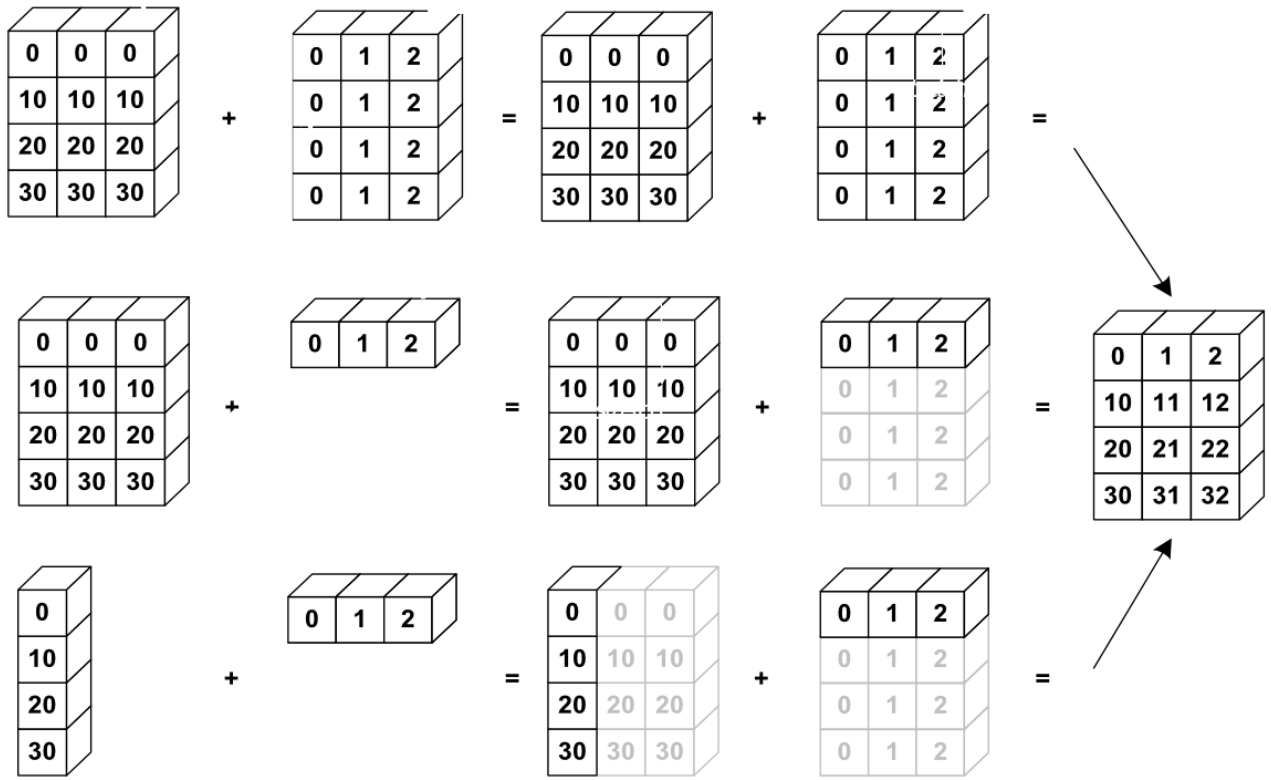
Array Broadcasting Rules

When operating on two arrays, NumPy compares their shapes element-wise. It starts with the trailing dimensions, and works its way forward. Two dimensions are compatible when

1. they are equal, or
2. one of them is 1

If these conditions are not met, a ValueError: frames are not aligned exception is thrown, indicating that the arrays have incompatible shapes. The size of the resulting array is the maximum size along each dimension of the input arrays.

Array Broadcasting Example



NumPy

SciPy

What is SciPy?

SciPy is a library of algorithms and mathematical tools built to work with NumPy arrays.

- statistics - *scipy.stats*
- optimization - *scipy.optimize*
- sparse matrices - *scipy.sparse*
- signal processing - *scipy.signal*
- etc.

Example: KS-test

Question: do two data samples come from the same distribution?

```
import scipy.stats as stats

# generate two data samples from different distributions
samp1 = stats.norm.rvs(loc=0., scale=1., size=100)
samp2 = stats.norm.rvs(loc=2., scale=1., size=100)

# perform ks test: null hypothesis is distributions are the same
D, pval = stats.ks_2samp(samp1, samp2)  # D=.58, pval=1.34e-15

# reject the null


# generate two data samples from the same distribution
samp1 = stats.norm.rvs(loc=0., scale=1., size=100)
samp2 = stats.norm.rvs(loc=0., scale=1., size=100)

# perform ks test
D, pval = stats.ks_2samp(samp1, samp2)  # D=.09, pval=.79

# fail to reject the null
```

Example: bootstrapped confidence interval

```
import numpy as np
import scipy.stats as stats

B = 1000
N = 100

# data array
arr = stats.norm.rvs(loc=np.pi, size=100)

# compute distribution of mean estimate
mean_distn = np.array([np.mean(arr[np.random.randint(N, size=N)])
                       for i in xrange(B)])

# 95% confidence interval [2.99, 3.39]
confidence_bounds = stats.mstats.mquantiles(mean_distn,
                                            prob=[.025, .975])
```

Example: k-Nearest Neighbor clustering

```
import sklearn.datasets
import sklearn.cluster

# boston house price data features are
#   neighborhood statisticsm and target is
#   median house value
boston = sklearn.datasets.load_boston()
k_means = sklearn.cluster.KMeans(n_clusters=10)
k_means.fit(boston.data)

# for each point, which cluster is it assigned?
k_means.labels_

# where is the center of each cluster?
k_means.cluster_centers_
```