Chapter 6, Data Analytic Tools

John M. Morrison

December 2, 2019

Contents

0 Introduction to the Toolkit

To install, we recommend you use the Anaconda package manager for Python. Complete instructions can be found at `https://docs.anaconda.com/anaconda/install/`. You will want to make sure you have these things.

1. `ipython` This is an enhanced Python interactive shell.
2. `numpy` This is a high-performance library for matrix calculations
3. `matplotlib` This is a library that can produce mathematical plots. They can be viewed locally on your machine, or it can generate graphical files on a server, which can be viewed off your account's web presence (`public_html`).
4. `pandas` This is a powerful tool that supports R-style data frames.

1 Creating Numpy Arrays

To see if numpy is working correctly, open `ipython` and do this import.

```
Python 3.6.8 |Anaconda, Inc.| (default, Dec 29 2018, 19:04:46)
Type 'copyright', 'credits' or 'license' for more information
```

```
IPython 7.2.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import numpy as np
In [2]:
```

If there is no error, you are good to go. Numpy supports the `ndarray`, or n–dimensional array object. These arrays are fixed-size mutable objects. They are *homogeous*; to wit, all objects must be of the same type. The type of objects present in an `ndarray` is called its `dtype`; this is short for "data type." Let us create an array and explore.

```
In [2]: x = np.array([0,0,0,0])

In [3]: x.dtype
Out[3]: dtype('int64')
In [4]: y = np.array([[0,0],[0,0]], dtype="float64")

In [5]: y
Out[5]:
array([[ 0.,  0.],
       [ 0.,  0.]])
```

In the first instance, Numpy saw the 0s and said, "Oh, integers." Note that these integers are 64 bit integers in two's complement notation. It auto-detected and assigned a default `dtype`.

In the second instance, Numpy was told the `dtype` was `float64` and cast the zeroes appropriately. It also saw list with two lists inside and made a 2×2 array. You have a choice when indexing into a two dimensional array.

```
In [6]: y[0,0]
Out[6]: 0.0

In [7]: y[0][0]
Out[7]: 0.0
```

The shape of an array is maintained as a tuple

```
In [8]: y.shape
Out[8]: (2, 2)

\section{Arithmetic, Functions, and Numpy Arrays}
In [9]: x.shape
Out[9]: (4,)
```

To fill make an array with nothing but zeroes you can do this.

```
In [10]: x = np.zeros(36)
In [11]: x
Out[11]:
array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.])
```

You can also specify the data type.

```
In [12]: y
Out[12]:
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

Watch the array get reshaped two ways.

```
In [13]: y.reshape(6,6)
Out[13]:
array([[0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0]])

In [14]: y.reshape(9,4)
Out[14]:
array([[0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0]])
```

You can make arrays with any number of dimensions; just specify a tuple. We also demonstrate the `fill` method.

```
In [15]: z
Out[15]:
array([[[ 0.,  0.,  0.],
        [ 0.,  0.,  0.],
        [ 0.,  0.,  0.],
```

```
        [ 0.,  0.,  0.]],

       [[ 0.,  0.,  0.],
        [ 0.,  0.,  0.],
        [ 0.,  0.,  0.],
        [ 0.,  0.,  0.]]])

In [16]: z.fill(5)

In [16]: z
Out[25]:
array([[[ 5.,  5.,  5.],
        [ 5.,  5.,  5.],
        [ 5.,  5.,  5.],
        [ 5.,  5.,  5.]],

       [[ 5.,  5.,  5.],
        [ 5.,  5.,  5.],
        [ 5.,  5.,  5.],
        [ 5.,  5.,  5.]]])
```

Exercises You should create some array so you can test the functions speced below.

1. If `x` is an array, what is `x.T`? Make sure you try this on a two-dimensional array.
2. Write a function `isSquare(x)` which tells if an two-dimensional array passed to it is a square array.
3. Write a function `swapRows(x, j, k)` that will swap the `j`th and `k`th rows in an array, and which throws a `ValueError` if the rows specified are out of bounds.
4. Write a function `swapCols(x, j, k)` that will swap the `j`th and `k`th columns in an array, and which throws a `ValueError` if the columns specified are out of bounds.
5. What does `np.eye(3)` produce?

2 Arithmetic on Numerical Arrays

Let us look at the behavior of numerical arrays when presented with arithmetic operators. We begin by making two 3×3 arrays.

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

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

In [3]: x
Out[3]:
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])

In [4]: y = np.eye(3)

In [5]: y
Out[5]:
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

Now let us play with arithmetic.

```
In [6]: x + y
Out[6]:
array([[  2.,   2.,   3.],
       [  4.,   6.,   6.],
       [  7.,   8.,  10.]])
```

The arrays were added element by element.

Programming Exercises Do these NOW!

1. What happens if you attempt to add arrays of different sizes?
2. What happens if you try to add an array to a number?

Apparently, multiplying an array by a number just multiplies every entry by that number.

```
In [7]: x + 5*y
Out[7]:
array([[  6.,   2.,   3.],
       [  4.,  10.,   6.],
       [  7.,   8.,  14.]])
```

If you are a linear algebra fan, start cringing now. However, you will see later that all will end well

```
In [8]: x*y
Out[8]:
```

```
array([[ 1.,  0.,  0.],
       [ 0.,  5.,  0.],
       [ 0.,  0.,  9.]])
```

If you multiply arrays, they multiply entrywise. Let's inject some power.

```
In [9]: x**3
Out[9]:
array([[  1,   8,  27],
       [ 64, 125, 216],
       [343, 512, 729]])
```

Now we will apply a function. All of the math functions have `np.` analogs that apply the entry-by-entry.

```
In [10]: np.log10(x)
Out[10]:
array([[ 0.        ,  0.30103   ,  0.47712125],
       [ 0.60205999,  0.69897   ,  0.77815125],
       [ 0.84509804,  0.90308999,  0.95424251]])

In [11]: np.sin(x)
Out[11]:
array([[ 0.84147098,  0.90929743,  0.14112001],
       [-0.7568025 , -0.95892427, -0.2794155 ],
       [ 0.6569866 ,  0.98935825,  0.41211849]])
```