2d arrays in NumPy

Timothy E. Holy

January 24, 2020

Unlike lists, NumPy supports arrays that are more than one dimension. There are several fundamental reasons you might want this. One of the easiest to understand is to ask yourself, how would you represent an *image* on the computer?

First, a quick convention: below, when I say `>>> some.python.command(arg)`, I am typing it at the Python prompt. (That's what I mean by the `>>>`.) If I then just show some text, that's Python's output. For example:

```
>>> type(3.0)
<class 'float'>
```

I typed the `type(3.0)`, and Python responded with `"<class 'float'>"`. Depending on how you're running Python, the output might be slightly different.

Images as 2d arrays

With that out of the way, let's load an image:

```
>>> from skimage import data
>>> img = data.moon()
```

Here, `skimage` is a module for image processing in Python. The `data` sub-module has some sample images in it; we decided to use one called `moon`.

Let's check that indeed we have an image:

```
>>> import matplotlib.pyplot as plt
>>> plt.imshow(img, cmap='gray')   # use a grayscale colormap
```

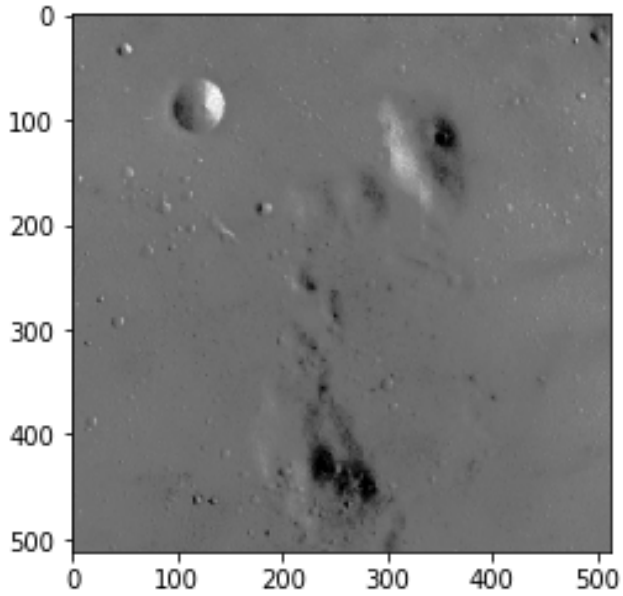
If you're following along, here's what I got in Spyder:

```
In [1]: from skimage import data

In [2]: img = data.moon()

In [3]: import matplotlib.pyplot as plt

In [4]: plt.imshow(img, cmap='gray')
Out[4]: <matplotlib.image.AxesImage at 0x7fbfaf3a25f8>
```



This is a zoom in of the surface of the moon.

OK, so what is `img`?

```
>>> type(img)
<class 'numpy.ndarray'>
```

Our old friend the NumPy `ndarray`! Let's look at the values:

```
>>> img.shape
(512, 512)
>>> img
array([[116, 116, 122, ...,  93,  96,  96],
       [116, 116, 122, ...,  93,  96,  96],
       [116, 116, 122, ...,  93,  96,  96],
       ...,
       [109, 109, 112, ..., 117, 116, 116],
       [114, 114, 113, ..., 118, 118, 118],
       [114, 114, 113, ..., 118, 118, 118]], dtype=uint8)
```

This means the image is 512×512 pixels.

The `dtype=uint8` means that the values are integers that range from 0..255. (Don't worry too much about the exact details for

now.)

When you work with images, you're really just working with NumPy arrays, and everything we're about to do works even if you have a different kind of data stored in the array. We'll try a tiny 2d array so we can see what happens and then you can try similar things with the image.

Getting single elements

Here's a very small 2d array:

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

Just like `a[0]` gets the first element of a list, `A[0, 0]` gets the upper-left corner of this 2d array:

```
>>> A[0, 0]
1
>>> A[1, 0]     # 2nd row, 1st column
4
>>> A[0, 1]     # 1st row, 2nd column
2
```

Exercise: get the upper-left pixel value of `img`

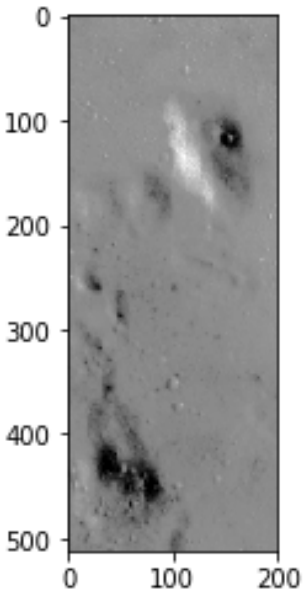
Getting a range

Numpy lets you get a range of values, again just like a list:

```
>>> A[0, 0:2]       # First row, first two items
array([1, 2])
>>> A[0:2, 0]       # First two rows, first column
array([1, 4])
>>> A[:, 0:2]       # Plain `:` means "the whole thing," in this case all rows
array([[1, 2],
       [4, 5]])
>>> A[:, [0, 2]]    # First and third columns
array([[1, 3],
       [4, 6]])
```

You can do this with the image too:

```
In [6]: plt.imshow(img[:,200:400], cmap='gray')
Out[6]: <matplotlib.image.AxesImage at 0x7fbfacad7898>
```



We grabbed all the "rows" of the image, but only a range of "columns" near the middle of the image.

Setting values

We can change the values in a NumPy array:

```
>>> A
array([[1, 2, 3],
       [4, 5, 6]])
>>> A[0, 1] = -7       # this changes the value to -7
>>> A
array([[ 1, -7,  3],
       [ 4,  5,  6]])
```

Exercise: Turn a 50×100 block in the upper left corner of `img` white. (255 is the value of "white".) Hint: you can use ranges when setting values, just like when you're getting values.

Another use for 2d arrays: matrix multiplication

2d arrays are also essential for matrix operations. Using the original `A` we defined above,

```
>>> b = np.array([10, 20, 30])
>>> A.dot(b)
array([140, 320])
```

(You'll get something different if your `A` has that modification where you set a value to `-7`.)

You can verify that this is the result you'd expect from multiplying the `A` matrix times the `b` vector.

Exercise: what happens if you write `A*b`? Can you guess what it's doing?

2d arrays as "lists of lists"

You can create a list that contains lists:

```
>>> lst = [[1, 2, 3], [4, 5, 6, 7]]
```

Note that this is a standard Python list, not a NumPy array. You can index it:

```
>>> lst[0]
[1, 2, 3]
```

and then index the result:

```
>>> lst[0][1]
2
```

Make sure you understand what's happening above: mentally read it like `(lst[0])[1]`, meaning it's like we defined

```
>>> item = lst[0]
>>> item[1]
```

You can do the same thing with a NumPy array:

```
>>> A[0][1]
2
```

However, only NumPy supports the `[i, j]` syntax:

```
>>> A[0, 1]
2
>>> lst[0, 1]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: list indices must be integers or slices, not tuple
```

Exercise: What happens if you turn `lst` into a `np.array`? Does `lst[0, 1]` work? If not, why do you think this is? What if you make both sub-lists be of the same length? (E.g., delete the `7`.)