NUMPY

9.17.2018

PROBLEM SET 1

* is posted.

* check canvas and/or github

* please make sure you understand how to use jupyter to run the notebook

* my office hours: MW 1:30-3pm, NHB 3.134

PYTHON + NUMBERS

- Plain python = bad with lots of numbers
  - Because it’s slow
  - & annoying w/ multidimensional arrays (have fun on the homework)

NUMPY

Travis Oliphant

- **import numpy as np**

- numpy has the **ndarray** (n-dimensional array) class

  - much easier (binary funcs & ufuncs)

  - much faster (sequential memory)

- *but:* arrays have fixed size, fixed type

SIGH MATLAB

- if you know *sigh* MATLAB, then a lot of numpy will be familiar

- you should read “numpy for [*sigh*] MATLAB users”: https://docs.scipy.org/doc/numpy/user/numpy-for-matlab-users.html

READING!

* Before next Monday (9/24),

* read Ch. 5 of “Inferential Thinking”: https://www.inferentialthinking.com/chapters/05/Sequences

* read Ch. 2 of “Python Data Science Handbook”: https://jakevdp.github.io/PythonDataScienceHandbook/02.00-introduction-to-numpy.html

NDARRAYS

* Let’s say **arr** is an ndarray

* arr.shape => array dimensions

  * shape is not a function, **don’t** try to use arr.shape()!

* arr.dtype => array type (int, float, etc)

  * dtype is also not a function!

CREATING NDARRAYS

```
* from a list:
  >>> np.array(some_list)

* a range:
  >>> np.arange(length)

* all zeros or ones:
  >>> np.zeros((nrows,ncols,…))
  >>> np.ones((nrows,ncols,…))
```

CREATING NDARRAYS

```
* linearly spaced:
  >>> np.linspace(start, end, steps)

* logarithmically spaced:
  >>> np.logspace(start, end, steps)
```

CREATING NDARRAYS

```
* random (uniform):
  >>> np.random.rand(nrows, ncols, …)

* random (gaussian):
  >>> np.random.randn(nrows, ncols, …)
```

BASIC OPERATIONS

* arithmetic with ndarrays and numbers

*
  ```
  >>> arr1 + 1
  >>> arr1 - 1
  >>> arr1 * 2.2
  >>> arr1 / 7
  >>> arr1 ** 2
  ```

BASIC OPERATIONS

* arithmetic with ndarrays and ndarrays

*
  ```
  >>> arr1 + arr2
  >>> arr1 - arr2
  >>> arr1 * arr2 # elementwise product!
  >>> arr1 / arr2 # also elementwise
  >>> arr1 ** arr2 # weird
  ```

BASIC INDEXING

* indexing into ndarrays is fast and awesome

*
  ```
  >>> arr1.shape
  (100, 35, 3)

  >>> arr1[12,23,0]
  ```

BASIC INDEXING

-
  ```
  >>> arr1.shape
  (100, 35, 3)

  >>> arr1[:10].shape
  (???)
  ```

BASIC INDEXING

-
  ```
  >>> arr1.shape
  (100, 35, 3)

  >>> arr1[:10].shape
  (10, 35, 3)
  ```

BASIC INDEXING

* multiple dimensions can be sliced at the same time

* >>> arr1[:10,23:29,:2]

    * *(still has 3 dimensions)*

* >>> arr1[:10,23:29,0]

    * *(only has two dimensions!)*

BASIC INDEXING

* skip dimensions in indexing using :

* >>> arr1[:,23:29,:2]

* >>> arr1[:,:,1]

UFUNCS

```
* np.sin(arr), np.cos(arr), …

* np.exp(arr), np.log(arr), …

* np.sqrt(arr), np.abs(arr), …
```

THAT’S IT