NumPy

What is NumPy?

- Python's main scientific computing package
- Main object: N-dimensional array (`ndarray`)
- Lots of *compiled* operations (fast!)
  - Math
  - Logic
  - I/O
  - Linear algebra
  - Statistics
  - Randomization
  - …

`ndarray`

- Basis for most scientific computing packages
- Homogeneous type (typically numbers)
- Fixed size: changing the size of an `ndarray` creates a new one
  - Memory footprint *may* change dynamically, since elements of an `ndarray` can be arbitrary Python objects

`ndarray`

- Supports advanced operations on large amounts of data with less code and more efficiently
- Element-by-element operations (+, –, `*`, `...`) are default
- Each dimension is an *axis*
  - By convention, last two axes correspond to rows and columns of matrices

Creating arrays

- `np.array(sequence)` — copy elements of `sequence` to an array
  - Type of the array is deduced from element types in `sequence`
  - Optional argument `dtype` to specify the array type
  - Nested sequences of depth N are transformed into N-dimensional arrays
- `np.zeros(shape)`, `np.ones(shape)`, `np.full(shape, val)` — array of all-zeros/ones/val with fixed shape
  - Avoids growing size if elements are initially unknown but shape is known
  - `shape` is a tuple of axis sizes
- `np.empty(shape)` — array of arbitrary elements with fixed shape
- `np.zeros_like(a)`, `np.ones_like(a)`, `np.full_like(a)`, `np.empty_like(a)` — copy shape from a

Creating arrays

- `np.arange(start, stop, step)` — analogous to `range()`
  - All arguments may be floating points
  - E.g., `np.arange(0, 1, 0.1)`
  - Floating point precision makes exact number of resulting elements hard to predict
- `np.linspace(start, stop, num_elements)` — like `np.arange()`, but with fixed number of elements

Basic operations

- Arithmetic operators (+, –, *, /) are applied element-wise
- Matrix product is performed with `a @ b` or `a.dot(b)`
  - Not with *
  - Nobody uses @ notation
- Some operations (+=, –=, *=, /=) act in place

Unary and universal operations

- Many unary operations (`sum`, `max`, `min`, `cumsum`) are methods of `ndarray`
  - E.g., `a.max()`
  - Operate by default on “flattened” array
  - Optional argument `axis` indicates dimension along which to operate
- Universal operations (`exp`, `sin`, `cos`, `sqrt`) also operate element-wise

Indexing, slicing, and iterating

- One-dimensional arrays are indexed and sliced like sequences
- Multi-dimensional arrays have one index per axis
  - Entire index is given as a tuple
  - Each index can itself be a slice
  - `a[:10, 1]` — first ten elements of the second column
  - `a[-10:, :]` — last ten elements of all columns
  - Missing slices are treated as complete slices (`:`)
  - Use dots (`...`) to indicate all needed complete slices
    - E.g., `a[:3, 1:6, ..., 5:]`
- Iterating is done over first axis
  - Use `a.flat` to iterate element-wise

Linear algebra

- `np.linalg.inv(a)` — matrix inverse
- `np.eye(n)` — identity matrix of size `n`
- `np.trace(a)` — trace of `a` (sum of diagonal elements)
- `np.linalg.solve(a, b)` — solve `ax = b` for `x`
- `np.linalg.eig(a)` — egienvalues and eigenvectors of `a`

There’s much more to NumPy

- This barely covers NumPy’s quickstart tutorial!
- It’s impossible to learn all of NumPy’s functionality
- So how do you know when NumPy has the function you need?
  - Usually, if you are looping through an array, you can vectorize your code
  - If fancy indexing is not enough, then there might be a NumPy function for what you need