CSC 223 - Advanced Scientific Programming

Numpy

Numpy

- Numpy (Numerical Python) provides an interface, called an array, to operate on dense data buffers.
- Numpy arrays are at the core of most Python scientific libraries.

The Numpy Array Type

- The Numpy array type is similar to a Python list, but all elements must be the same type.
- The numpy `array` function is used to construct arrays
- Example: construct from a list

  ```
  import numpy as np
  np.array([1, 2, 3, 4])
  ```

- Example: set the element type

  ```
  import numpy as np
  np.array([1, 2, 3, 4], dtype=’float32’)
  ```

Creating Numpy Arrays

- `zeros`: create an array of zeros
- `ones`: create an array of ones
- `full`: create an array filled with a specified value
- `arange`: create an array from a Python range
- `linspace`: create an array of evenly spaced values
- `random.random`: create an array of random values between 0 and 1

Array Creation Examples

- Create an array with even numbers from 0 to 10

  ```
  np.arange(0, 10, 2)
  ```

- Create a 3×3 array of random values

  ```
  np.random.random((3,3))
  ```

- Create a 2×5 array filled with integers with value 7

  ```
  np.full((2,5), 7, dtype=int)
  ```

Some Array Data Types

- `bool_`: Boolean stored as a byte
- `int8`: Byte (-128 to 127)
- `int16, int32, int64`: Integers
- `float16, float32, float64`: Floating point numbers
- `complex64, complex128`: Complex numbers

Basic Array Manipulations

- Attributes of arrays
- Indexing arrays
- Slicing arrays
- Reshaping arrays
- Joining and splitting arrays

Array Attributes

- `ndim`: the number of dimensions
- `shape`: the size of each dimension
- `size`: the total size of the array
- `dtype`: the data type of the array
- `itemsize`: the size in bytes of each element
- `nbytes`: the total size in bytes

Array Indexing

- Numpy arrays can be indexed like Python lists.
- Numpy arrays with multiple dimensions can be indexed with a tuple
- Example:

```
x = np.array([[1,2,3],
              [4,5,6],
              [7,8,9]])
x[0,0] # get 1
x[1,2] # get 6
x[2,-1] # get 9
```

Array Slicing

- Numpy arrays can be sliced like Python lists:

  ```
  x[start:stop:step]
  ```

- Numpy arrays with multiple dimensions can be sliced in each dimension
- Example:

  ```
  x = np.array([[1,2,3],
                [4,5,6],
                [7,8,9]])
  >>> x[0:2, 0:2]
  array([[1,2],
         [4,5]])
  >>> x[2, :] # get the third row
  array([7,8,9]])
  ```

Copy an Array

- Array slices return *views* (shallow copies) into arrays
- The `copy` method can be used to create a deep copy of an array

Array Manipulation

- `reshape`: change the shape of an array
- `concatenate`: concatenate arrays
- `hstack`: horizontally stack arrays
- `vstack`: vertically stack arrays
- `split`: split an array
- `hsplit`: split an array horizontally
- `vsplit`: split an array vertically

Universal Functions

- Numpy has universal functions (UFuncs) that can perform operations on entire arrays
- Some ufuncs:
  - `absolute`
  - `add` or `+`
  - `divide` or `/`
  - `multiply` or `*`
  - `power` or `**`
  - `subtract` or `-`
- For arrays of the same size, binary operations are performed element-wise

Aggregations

- `all`: check if all elements are true
- `any`: check if any elements are true
- `argmax`: index of maximum element
- `argmin`: index of minimum element
- `max`: max value
- `mean`: mean value
- `median`: median
- `min`: min value
- `percentile`: rank-based statistics
- `prod`: product of elements
- `std`: standard deviation
- `sum`: sum of elements
- `var`: variance

Multi-Dimensional Aggregations

- In a multi-dimensional array, aggregations can be performed along a row or a column.
- The `axis` argument specifies the axis along which the aggregate is computed
- Example:

```
x = np.array([[1,2,3],
              [4,5,6]])
>>> sum(x, axis=0)
array([5, 7, 9])
>>> sum(x, axis=1)
array([6, 15])
```

Sorting Arrays

- The `sort` function returns a sorted array: `sort(x)`
- The `sort` method does an in place sort: `x.sort()`
- `argsort` returns indices of the sorted elements
- Multidimensional arrays can be sorted along axes: `sort(x, axis=1)`

Broadcasting

- Broadcasting allows ufuncs to be applied to arrays of different sizes.
- Rules of broadcasting:
  1. If the two arrays differ in dimensions, the shape of the one with fewer dimensions is padded with ones on its left side.
  2. If the shape of the two arrays does not match any dimension, the array with shape equal to one in that dimension is stretched to match the other shape.
  3. If sizes disagree in any dimension and neither is equal to 1, then an error is raised.

Broadcasting Example

```
>>> A = np.ones( (2,3) )
>>> b = np.arange(3)
>>> A + b
array([[ 1., 2., 3.],
       [ 1., 2., 3.]])
# A: shape (2,3)
# b: shape (3,) -> (1,3) -> (2,3)
```

Boolean Arrays

- Comparison operators
  - `equal` or `==`
  - `not_equal` or `!=`
  - `less` or `<`
  - `less_equal` or `<=`
  - `greater` or `>`
  - `greater_equal` or `>=`
- Boolean operators
  - `bitwise_and` or `&`
  - `bitwise_or` or `|`
  - `bitwise_xor` or `^`
  - `bitwise_not` or `~`

Boolean Operation Example

```
x = np.random.random( (10,2) )

# number of entries greater than 0.5
np.sum(x > 0.5)

# count entries between values
np.sum( (x > 0.25) & (x < 0.75) )
```

Boolean Masks

- An array can be indexed with a boolean expression
- The result is a one dimensional array with the values that satisfy the expression
- Example:

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

Fancy Indexing

- Fancy indexing allows indexing of an array by passing in an array of indices
- Fancy indexing can be combined with slicing
- Example:

```
>>> x = np.array([[1,2,3],
                  [4,5,6],
                  [7,8,9]])
>>> row = np.array([0, 1, 2])
>>> col = np.array([2, 1, 0])
>>> x[row, col]
array([3, 5, 7])
```