Arrays are the central datatype introduced in the NumPy and SciPy packages. (Technically, the objects are of type ndarray, presumably for “n-dimensional array”.) The array interface is accessible either via import numpy or via import scipy. Arrays are similar in some respects to Python lists, but are multidimensional, homogeneous in type, and support compact and efficient array-level manipulations. Documentation can be found online at www.scipy.org/Documentation, which also includes links to NumPy Examples (sample usage for many functions) and NumPy for MATLAB Users, if you’re so inclined.

Constructing arrays

- `numpy.array(alist)`: construct an n-dimensional array from a Python list (all elements of list must be of same length)
  
  ```
  a = numpy.array([[1,2,3],[4,5,6]])
  b = numpy.array([i*i for i in range(100) if i%2==1])
  c = b.tolist()                            # convert array back to Python list
  ```

- `numpy.zeros(shape, dtype=float)`: construct an n-dimensional array of the specified shape, filled with zeros of the specified dtype; e.g.,
  
  ```
  a = numpy.zeros(100)                      # a 100-element array of float zeros
  b = numpy.zeros((2,8), int)               # a 2x8 array of int zeros
  c = numpy.zeros((N,M,L), complex)         # a NxMxL array of complex zeros
  ```

- `numpy.ones(shape, dtype=float)`: construct an n-dimensional array of the specified shape, filled with ones of the specified dtype; e.g.,
  
  ```
  a = numpy.ones(10, int)                   # a 10-element array of int ones
  b = numpy.pi * numpy.ones((5,5))          # a useful way to fill up an array with a specified value
  ```

- `numpy.eye(shape, dtype=float)`: construct an n-dimensional identity matrix (1's on diagonal)
  
  ```
  id = numpy.eye(10,10, int)                           # 10x10 identity matrix (1's on diagonal)
  offdiag = numpy.eye(10,10,1)+numpy.eye(10,10,-1)     # off diagonal elements = 1
  ```

- `numpy.linspace(start, stop, num_elements)`: create array of equally-spaced points based on specified number of points
  
  ```
  a = numpy.linspace(start, stop, num_elements) # create array of equally-spaced points based on specified number of points
  ```

- `numpy.random.random((100,100))`: 100x100 array of floats uniform on [0.,1.)
  
  ```
  a = numpy.random.random((100,100))
  b = numpy.random.randint(0,10, (100,)) # 100 random ints uniform on [0, 10), i.e., not including the upper bound 10
  c = numpy.random.standard_normal((5,5,5)) # zero-mean, unit-variance Gaussian random numbers in a 5x5x5 array
  ```

Indexing arrays

- Multidimensional indexing
  
  ```
  elem = a[i,j,k]         # equiv. to a[i][j][k] but presumably more efficient
  ```
"Negative" indexing (wrap around the end of the array)

last_elem = a[-1]  # the last element of the array

Arrays as indices

i = numpy.array((0,1,2,1))  # array of indices for the first axis
j = numpy.array((1,2,3,4))  # array of indices for the second axis
a[i,j]  # return array([a[0,1], a[1,2], a[2,3], a[1,4]])
b = numpy.array([True, False, True, False])
a[b]  # return array([a[0], a[2]]) since only b[0] and b[2] are True

Slicing arrays (extracting subsections)

- Slice a defined subblock:
  
  section = a[10:20, 30:40]  # 10x10 subblock starting at [10,30]

- Grab everything up to the beginning/end of the array:
  
  asection = a[10:, 30:]  # missing stop index implies until end of array
  bsection = b[10:, :30]  # missing start index implies until start of array

- Grab an entire column(s)
  
  x = a[: , 0]  # get everything in the 0th column (missing start and stop)
  y = a[: , 1]  # get everything in the 1st column

- Slice off the tail end of an array
  
  tail = a[:,-10:]  # grab the last 10 elements of the array
  slab = b[:,-10:]  # grab a slab of width 10 off the "side" of the array
  interior = c[1:-1, 1:-1, 1:-1]  # slice out everything but the outer shell

Element-wise functions on arrays

- Arithmetic operations
  
  c = a + b  # add a and b element-wise (must be same shape)
  d = e * f  # multiply e and f element-wise (NOT matrix multiplication)
  g = -h  # negate every element of h
  y = (x+1)%2  # swap 0's and 1's in binary array x
  z = w > 0.0  # return boolean array indicating which elements are > 0.0
  logspace = 10.***numpy.linspace(-6.0, -1.0, 50)  # 50 equally-spaced-in-log points between 1.e-06 and 1.e-01

- Trigonometric operations
  
  y = numpy.sin(x)  # sin of every element of x
  w = numpy.sin([i*i for i in range(100) if i%2==1])  # conversion from list to array as part of function application
  z = numpy.exp((0.+1.j) * theta)  # exp(i * theta) where i = sqrt(-1) = 0.+1.j

Summation of arrays

- Simple sums
  
  s = numpy.sum(a)  # sum all elements in a, returning a scalar
  s0 = numpy.sum(a, axis=0)  # sum elements along specified axis (>0), returning an array of remaining shape, e.g.,
  a = numpy.ones((10,20,30))
  s0 = numpy.sum(a, axis=0)  # s0 has shape (20,30)

- Averaging, etc.
  
  m = numpy.mean(a, axis)  # compute mean along the specified axis (over entire array if axis=None)
  s = numpy.std(a, axis)  # compute standard deviation along the specified axis (over entire array if axis=None)

- Cumulative sums
  
  s0 = numpy.cumsum(a, axis=0)  # cumulatively sum over 0 axis, returning array with same shape as a
  s0 = numpy.cumsum(a)  # cumulatively sum over 0 axis, returning 1D array of length shape[0]*shape[1]*...*shape[dim-1]

Various other useful functions and methods (see NumPy Examples at www.scipy.org/NumPy_Example_List_With_Doc)

Many of these work both as separate functions (numpy.blah(a)) as well as array methods (a.blah()).

- numpy.any(a): return True if any element of a is True
- numpy.all(a): return True if all elements of a are True
- `numpy.alltrue(a, axis)`: perform logical_and along given axis of `a`
- `numpy.append(a, values, axis)`: append values to `a` along specified axis
- `numpy.concatenate((a1, a2, ...), axis)`: concatenate tuple of arrays along specified axis
- `numpy.min(a, axis=None), numpy.max(a, axis=None)`: get min/max values of `a` along specified axis (global min/max if `axis=None`)
- `numpy.argmin(a, axis=None), numpy.argmax(a, axis=None)`: get indices of min/max of `a` along specified axis (global min/max if `axis=None`)
- `numpy.reshape(a, newshape)`: reshape `a` to `newshape` (must conserve total number of elements)
- `numpy.matrix(a)`: create matrix from 2D array `a` (matrices implement matrix multiplication rather than element-wise multiplication)
- `numpy.histogram, numpy.histogram2d, numpy.histogramdd`: 1-dimensional, 2-dimensional, and d-dimensional histograms, respectively
- `numpy.round(a, decimals=0)`: round elements of matrix `a` to specified number of decimals
- `numpy.sign(a)`: return array of same shape as `a`, with -1 where `a < 0`, 0 where `a = 0`, and +1 where `a > 0`
- `a.tofile(fid, sep='', format='''')`: write `a` to specified file `fid`, in either binary or ascii format depending on options
- `numpy.fromfile(file=, dtype=float, count=-1, sep='')`: read array from specified file (binary or ascii)
- `numpy.unique(a)`: return sorted unique elements of array `a`
- `numpy.where(condition, x, y)`: return array with same shape as `condition`, where values from `x` are inserted in positions where `condition` is True, and values from `y` where `condition` is False