EML4930/EML6934: Lecture 04

Introduction to NumPy and matrix operations

Charles Jekel

September 21, 2017

Quiz at the end of next class

There will be a quiz at the end of next class.

The quiz will be closed notes!

The quiz will be 3 short questions.

10 sample quiz questions are posted online. You should know how to answer each one of these questions. The 3 questions asked will be very similar to these 10 questions.

Concerns about HW?

The Fibonacci problem can be coded without the need of iterating through an index. If you are working with lists in Python you may not need the list index!

```
F = [0, 1, 1]
#    Calculate the next 20 items in the Fibonacci sequence
for i in range(20):
    F.append(F[-1]+F[-2])
```

Reference for this lecture

This lecture will cover chapter 2 of Python Data Science Handbook by VanderPlas.

`http://nbviewer.jupyter.org/github/jakevdp/PythonDataScienceHandbook/blob/master/notebooks/Index.ipynb`

NumPy - `http://numpy.org`

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities
- documentation `https://docs.scipy.org/doc/numpy-1.13.0/reference/index.html`

If you’re coming from MATLAB read

`https://docs.scipy.org/doc/numpy-dev/user/numpy-for-matlab-users.html`

Importing NumPy

```python
import numpy as np
```

This is the most standard form of how to import the numpy library.

What is a NumPy array?

NumPy ndarray()

A n-dimensional arrays of homogeneous data types.

- The *ndarray* object is the core of NumPy.
- *ndarray* has become the standard Python vector/matrix/tensor types.
- Elements of *ndarray* must have the same data type.
- Mathematical operations on *ndarray* are more efficient than built-in Python functions.
- Much of the code was created in compiled code for performance.
- Documentation `https://docs.scipy.org/doc/numpy-1.10.0/reference/arrays.ndarray.html`

NumPy array attributes - memory layout

The following attributes contain information about the memory layout of the array:

- ndarray.flags - Information about the memory layout of the array.
- ndarray.shape - Tuple of array dimensions.
- ndarray.strides - Tuple of bytes to step in each dimension when traversing an array.
- ndarray.ndim - Number of array dimensions.
- ndarray.data - Python buffer object pointing to the start of the arrays data.
- ndarray.size - Number of elements in the array.
- ndarray.itemsize - Length of one array element in bytes.
- ndarray.nbytes - Total bytes consumed by the elements of the array.
- ndarray.base - Base object if memory is from some other object.

NumPy array attributes - other attributes

The following attributes contain information about the memory layout of the array:

- ndarray.dtype - Data-type of the arrays elements.
- ndarray.T - Same as self.transpose(), except that self is returned if self.ndim < 2.
- ndarray.real - The real part of the array.
- ndarray.imag - The imaginary part of the array.
- ndarray.flat - A 1-D iterator over the array.
- ndarray.ctypes - An object to simplify the interaction of the array with the ctypes module.

NumPy array methods - array conversion 1 of 2

- ndarray.item(*args) - Copy an element of an array to a standard Python scalar and return it.
- ndarray.tolist() - Return the array as a (possibly nested) list.
- ndarray.itemset(*args) - Insert scalar into an array (scalar is cast to array's dtype, if possible) There must be at least 1 argument, and define the last argument as item.
- ndarray.tostring([order]) - Construct Python bytes containing the raw data bytes in the array.
- ndarray.tobytes([order]) - Construct Python bytes containing the raw data bytes in the array.
- ndarray.tofile(fid[, sep, format]) - Write array to a file as text or binary (default).
- ndarray.dump(file) - Dump a pickle of the array to the specified file.
- ndarray.dumps() - Returns the pickle of the array as a string.
- ndarray.astype(dtype[, order, casting, ...]) - Copy of the array, cast to a specified type.

NumPy array methods - array conversion 2 of 2

- ndarray.byteswap(inplace) - Swap the bytes of the array elements Toggle between low-endian and big-endian data representation by returning a byteswapped array, optionally swapped in-place.
- ndarray.copy([order]) - Return a copy of the array.
- ndarray.view([dtype, type]) - New view of array with the same data.
- ndarray.getfield(dtype[, offset]) - Returns a field of the given array as a certain type.
- ndarray.setflags([write, align, uic]) - Set array flags WRITEABLE, ALIGNED, and UPDATEIFCOPY, respectively.
- ndarray.fill(value) - Fill the array with a scalar value.

NumPy array methods - shape manipulation

- ndarray.reshape(shape[, order]) - Returns an array containing the same data with a new shape.
- ndarray.resize(new_shape[, refcheck]) - Change shape and size of array in-place.
- ndarray.transpose(*axes) - Returns a view of the array with axes transposed.
- ndarray.swapaxes(axis1, axis2) - Return a view of the array with axis1 and axis2 interchanged.
- ndarray.flatten([order]) - Return a copy of the array collapsed into one dimension.
- ndarray.ravel([order]) - Return a flattened array.
- ndarray.squeeze([axis]) - Remove single-dimensional entries from the shape of a.

Numpy array methods - methods that are also functions

- all
- any
- argmax
- argmin
- argpartition
- argsort
- choose
- clip
- compress
- copy
- cumprod
- cumsum
- diagonal
- imag
- max
- mean
- min
- nonzero
- partition
- prod
- ptp
- put
- ravel
- real
- repeat
- reshape
- round
- searchsorted
- sort
- squeeze
- std
- sum
- swapaxes
- take
- trace
- transpose
- var

NumPy array data types - 1 of 2

Type	Description
bool_	Boolean (True or False) stored as a byte
int_	Default integer type (normally either int64 or int32)
intc	Identical to C int (normally int32 or int64)
intp	Integer used for indexing (normally either int32 or int64)
int8	Byte (-128 to 127)
int16	Integer (-32768 to 32767)
int32	Integer (-2147483648 to 2147483647)
int64	Integer (-9223372036854775808 to 9223372036854775807)
uint8	Unsigned integer (0 to 255)
uint16	Unsigned integer (0 to 65535)
uint32	Unsigned integer (0 to 4294967295)
uint64	Unsigned integer (0 to 18446744073709551615)

NumPy array data types - 2 of 2

NumPy arrays support a larger variety of data types than the default Python!

Type	Description
float_	Shorthand for float64.
float16	Half precision: sign bit, 5 bits exponent, 10 bits mantissa
float32	Single precision: sign bit, 8 bits exponent, 23 bits mantissa
float64	Double precision: sign bit, 11 bits exponent, 52 bits mantissa
complex_	Shorthand for complex128.
complex64	Complex number, represented by two 32-bit floats
complex128	Complex number, represented by two 64-bit floats

NumPy array creation - from lists

Use

```
np.array(list)
```

to create an array from a Python list

Examples:

```
# one dimensional array shape (4,)
x = np.array([1, 2, 4, 6])

# two dimensional array shape (2,2)
y = np.array([[1 ,2], [2, 3]])

# three dimensional array - shape (2,2,2)
z = np.array([[[1 ,2], [2, 3]], [[1 ,2], [2, 3]]])
```

NumPy array creation - specify data type

You can specify the data type of the array using

```
np.array(list, dtype=)
```

Examples:

```
# one dimensional array - Byte
x = np.array([1, 2, 4, 6], dtype='int8')

# two dimensional array - Single precision
y = np.array([1, 2, 4, 6], dtype='float32')

# three dimensional array - Double precision
z = np.array([1, 2, 4, 6], dtype='float64')
```

NumPy array creation - display data type

You can view the data type with

```
ndarray.dtype)
```

Examples:

```
in : x.dtype
out: dtype('int8')

in : y.dtype
out: dtype('float32')

in : z.dytpe
out: dtype('float64')
```

Default data types follow python convention

If you include a decimal point NumPy array will default to double precision floating point.

Examples:

```
in : a = np.array(1); a.dtype
out: dtype('int32')

in : b = np.array(1.); b.dtype
out: dtype('float64')

in : c = np.array([1,1.]); c.dtype
out: dtype('float64')

in : d = np.array([1., 1, 1, 1, 1, 1, 1]); d.dtype
out: dtype('float64')
```

NumPy array creation - from scratch 1 of 3

These default to double precision floating point unless otherwise specified.

```
# Create a length-10 integer array filled with zeros
np.zeros(10, dtype=int)

# Create a 3x5 floating-point array filled with 1s
np.ones((3, 5), dtype=float)

# Create a 3x5 array filled with 3.14
np.full((3, 5), 3.14)

# Create an array filled with a linear sequence
# Starting at 0, ending at 20, stepping by 2
np.arange(0, 20, 2)
```

NumPy array creation - from scratch 2 of 3

```
# Create an array of five values evenly spaced between 0 and 1
np.linspace(0, 1, 5)

# Create a 3x3 array of uniformly distributed
# random values between 0 and 1
np.random.random((3, 3))

# Create a 3x3 array of normally distributed random values
# with mean 0 and standard deviation 1
np.random.normal(0, 1, (3, 3))

# Create a 3x3 array of random integers in the interval [0, 10)
np.random.randint(0, 10, (3, 3))

# Create a 3x3 identity matrix
np.eye(3)
```

NumPy array creation - from scratch 3 of 3

```
# Create an uninitialized array of three integers
# The values will be whatever happens to already exist at that
# memory location
np.empty(3)
```

Seeding NumPy pseudorandom number generator

Use seed to ensure that the arrays are the same for each time the code is run.

```
# seed for reproducibility
np.random.seed(0)

# One-dimensional array
x1 = np.random.randint(10, size=6)

# Two-dimensional array
x2 = np.random.randint(10, size=(3, 4))

# Three-dimensional array
x3 = np.random.randint(10, size=(3, 4, 5))
```

NumPy array - useful attributes

```python
# the number of dimensions
print("x3 ndim: ", x3.ndim)

# the size of each dimension of the array
print("x3 shape:", x3.shape)

# the total size of the array
print("x3 size: ", x3.size)

# the item size (in bytes) of each array element
print("itemsize:", x3.itemsize, "bytes")

# the total size (in bytes) of the array
print("itemsize:", x3.nbytes, "bytes")
```

NumPy array indexing - one dimensional

One dimensional arrays index just like lists

```
x = np.array([2, 3, 4, 9])

# print the first item
print(x[0])

# print the second item
print(x[1])

# print the last item
print(x[-1])
```

NumPy array indexing - two dimensional

Two dimensional arrays also index just like lists!

```
y = np.array([[1,2], [3,4]])

# print 1 from y
print(y[0,0])

# print 2 from y
print(y[0,1])

# print 3 from y
print(y[1,0])

# print 4 from y
print(y[1,1])
```

NumPy array slicing - just like list slicing!

```
in : x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
in : x[:5]  # first five elements
out: array([0, 1, 2, 3, 4])

in : x[5:]  # elements after index 5
out: array([5, 6, 7, 8, 9])

in : x[4:7]  # middle subarray
out: array([4, 5, 6])

in : x[::2]  # every other element
out: array([0, 2, 4, 6, 8])

in : x[1::2]  # every other element, starting at index 1
out: array([1, 3, 5, 7, 9])

in : x[::-1]  # all elements, reversed
```

NumPy array - rows and columns of

```
in : x = np.array([[12,  5,  2,  4],
                   [ 7,  6,  8,  8],
                   [ 1,  6,  7,  7]])

# first column of x
x[:,0]

# second column of x
x[:,1]

# third row of x
x[2,:]

# first row of x
x[0,:]
```

NumPy array - assigning individual values

```
in : x = np.array([[12,  5,  2,  4],
                   [ 7,  6,  8,  8],
                   [ 1,  6,  7,  7]])

x[0,0] = 0
x[0,1] = 1
x[0,2] = 2
x[0,3] = 3
print(x)

out: array([[ 0,  1,  2,  3],
            [ 7,  6,  8,  8],
            [ 1,  6,  7,  7]])
```

NumPy arrays pass by reference!

use copy() if you want a copy of the data!

```
a = np.ones(3)
b = a
c = a.copy()

a *= 7
print('a:', a)
print('b:', b)
print('c:', c)
```

NumPy aggregration functions

Function	NaN-safe	Description
np.sum	np.nansum	Compute sum of elements
np.prod	np.nanprod	Compute product of elements
np.mean	np.nanmean	Compute median of elements
np.std	np.nanstd	Compute standard deviation
np.var	np.nanvar	Compute variance
np.min	np.nanmin	Find minimum value
np.max	np.nanmax	Find maximum value
np.argmin	np.nanargmin	Find index of minimum value
np.argmax	np.nanargmax	Find index of maximum value
np.median	np.nanmedian	Compute median of elements
np.percentile	np.nanpercentile	Compute rank-based statistics of elements
np.any	N/A Evaluate	whether any elements are true
np.all	N/A Evaluate	whether all elements are true

NumPy useful functions

```
np.abs(x) # absolute value

# trig functions
np.sin(x)
np.cos(x)
np.tan(x)
np.arcsin(x)
np.arccos(x)
np.arctan(x)


np.exp(x)  # e^x
np.log(x)  # natural log
np.log2(x) # log base 2
np.log10(x)# log base 10
```

NumPy arrays - arithmetic operators

Operator	**Description**
+	Addition (e.g., 1 + 1 = 2)
-	Subtraction (e.g., 3 - 2 = 1)
-	Unary negation (e.g., -2)
*	Multiplication (e.g., 2 * 3 = 6)
/	Division (e.g., 3 / 2 = 1.5)
//	Floor division (e.g., 3 // 2 = 1)
**	Exponentiation (e.g., 2 3 = 8)
%	Modulus/remainder (e.g., 9 % 4 = 1)

NumPy arrays - element wise math

```
in :
a = np.array([2, 4, 6, 8])
b = np.array([3, 4, 5, 6])

print(a+b)
print(a-b)
print(a*b)
print(a**b)

out:
[ 5  8 11 14]
[-1  0  1  2]
[ 6 16 30 48]
[     8    256   7776 262144]
```

NumPy arrays - element wise math

```
in :
a = np.array([[2, 4],[6, 8]])
b = np.array([[3, 4],[5, 6]])
print(a+b)
print(a-b)
print(a*b)
print(a**b)

out:
[[ 5  8]
 [11 14]]
[[-1  0]
 [ 1  2]]
[[ 6 16]
 [30 48]]
[[     8    256]
 [  7776 262144]]
```

NumPy arrays - dot product

Use np.dot(a,b) to take the dot product of arrays a and b

```
in :
a = np.array([2, 4, 6, 8])
b = np.array([3, 4, 5, 6])
print(np.dot(a,b))
```

```
out: 100
```

NumPy arrays - dot product for matrix multiplication

The dot product of two dimensional arrays is matrix multiplication

```
in :
x = np.array([[4, 5, 2],
              [7, 7, 1],
              [9, 1, 7]])
y = np.eye(3)

print(np.dot(x,y))

out:
[[4, 5, 2],
 [7, 7, 1],
 [9, 1, 7]])
```

NumPy linear algebra - decomposition

function	**Description**
np.linalg.cholesky(a)	Cholesky decomposition
np.linalg.qr(a[,mode])	qr factorization
np.linalg.svd(a[, full_matrices, compute_uv])	Singular value decomposition

NumPy linear algebra - eigenvalues

```
in :
w, v = np.linalg.eig(np.diag((1, 2, 3)))

print(w)
print(v)

out:
array([ 1.,  2.,  3.])
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

NumPy linear algebra - norms and other numbers

Function	**Description**
np.linalg.norm(x)	Matrix or vector norm.
np.linalg.cond(x)	Compute the condition number of a matrix.
np.linalg.det(a)	Compute the determinant of an array.
np.linalg.matrix_rank(M)	Return matrix rank of array using SVD method.
np.linalg.slogdet(a)	Sign and ln of the determinant of an array.
np.trace(a)	Return the sum along diagonals of the array.

NumPy linear algebra - solving equations and inverting matrices

Function	Description
np.linalg.solve(a, b)	Solve a linear matrix equation
np.linalg.tensorsolve(a, b[, axes])	Solve the tensor equation a x = b for x
np.linalg.lstsq(a, b[, rcond])	least-squares solution to a linear equation
np.linalg.inv(a)	multiplicative) inverse of a matrix
np.linalg.pinv(a[, rcond])	(Moore-Penrose) pseudo-inverse of a matri
np.linalg.tensorinv(a[, ind])	inverse of an N-dimensional array

Don't invert arrays! Use solve!

np.linalg.solve automatically uses parallel processing if it suspects using multiple threads will be faster!

```
F = np.array([[2.0, 3.0],
              [-2.0,9.0]])
c = np.array([ 12.9,  12.3])

# solve the equation Fa = c for a
a = np.linalg.solve(F,c)

# check the solution
print(np.dot(F,a))
```