COMPSCI 590N
Lecture 5: NumPy 1

Roy J. Adams

College of Information and Computer Sciences
University of Massachusetts Amherst

Outline

What is NumPy?

Numerical Python (NumPy) is the backbone of the Python scientific programming stack:

- Provides multidimensional arrays (tensors).

What is NumPy?

Numerical Python (NumPy) is the backbone of the Python scientific programming stack:

- Provides multidimensional arrays (tensors).
- Most functions written in C (more efficient).

What is NumPy?

Numerical Python (NumPy) is the backbone of the Python scientific programming stack:

- Provides multidimensional arrays (tensors).
- Most functions written in C (more efficient).
- Many useful functions for speeding up loops over array elements (vectorization).

What is NumPy?

Numerical Python (NumPy) is the backbone of the Python scientific programming stack:

- Provides multidimensional arrays (tensors).
- Most functions written in C (more efficient).
- Many useful functions for speeding up loops over array elements (vectorization).

What is NumPy?

Numerical Python (NumPy) is the backbone of the Python scientific programming stack:

- Provides multidimensional arrays (tensors).
- Most functions written in C (more efficient).
- Many useful functions for speeding up loops over array elements (vectorization).

Typical import convention:

```
import numpy as np
```

The Numpy Array

The core of NumPy is the Array type:

The Numpy Array

The core of NumPy is the Array type:

```
>>> import numpy as np
>>> a = np.array([0, 1, 2, 3])
>>> a
array([0, 1, 2, 3])
```

The Numpy Array

The core of NumPy is the Array type:

```
>>> import numpy as np
>>> a = np.array([0, 1, 2, 3])
>>> a
array([0, 1, 2, 3])
```

- Flexible structure good for storing typical scientific data:

The Numpy Array

The core of NumPy is the Array type:

```
>>> import numpy as np
>>> a = np.array([0, 1, 2, 3])
>>> a
array([0, 1, 2, 3])
```

- Flexible structure good for storing typical scientific data:
  - Numerical data matrix

The Numpy Array

The core of NumPy is the Array type:

```
>>> import numpy as np
>>> a = np.array([0, 1, 2, 3])
>>> a
array([0, 1, 2, 3])
```

- Flexible structure good for storing typical scientific data:
  - Numerical data matrix
  - Image pixel values

The Numpy Array

The core of NumPy is the Array type:

```
>>> import numpy as np
>>> a = np.array([0, 1, 2, 3])
>>> a
array([0, 1, 2, 3])
```

- Flexible structure good for storing typical scientific data:
  - Numerical data matrix
  - Image pixel values
  - 3-d images such as MRI scan data

The Numpy Array

The core of NumPy is the Array type:

```
>>> import numpy as np
>>> a = np.array([0, 1, 2, 3])
>>> a
array([0, 1, 2, 3])
```

- Flexible structure good for storing typical scientific data:
  - Numerical data matrix
  - Image pixel values
  - 3-d images such as MRI scan data
- Mutable contents, but not size.

The Numpy Array

The core of NumPy is the Array type:

```
>>> import numpy as np
>>> a = np.array([0, 1, 2, 3])
>>> a
array([0, 1, 2, 3])
```

- Flexible structure good for storing typical scientific data:
  - Numerical data matrix
  - Image pixel values
  - 3-d images such as MRI scan data
- Mutable contents, but not size.
- In general, no mixed types.

Constructing Arrays

Arrays can be created manually:

Constructing Arrays

Arrays can be created manually:

```
>>> a = np.array([1, 2]) # 1-D
>>> a
array([1, 2])
>>> b = np.array([[1,2],[3,4]]) # 2-D
>>> b
array([[1, 2],
       [3, 4]])
>>> c = np.array([[[1,2],[3,4]],[[5,6],[7,8]]]) # 3-D
>>> c
array([[[1, 2],
        [3, 4]],

       [[5, 6],
        [7, 8]]])
```

Constructing Arrays

Many functions exist to create NumPy Arrays:

Constructing Arrays

Many functions exist to create NumPy Arrays:

```
# Create a range
>>> np.arange(5)
array([0, 1, 2, 3, 4])
>>> np.arange(0,10,2) # start, end, step size
array([0, 2, 4, 6, 8])

# Or specify a number of points
>>> np.linspace(0,1,5) # start, end, number of points
array([ 0.  ,  0.25,  0.5 ,  0.75,  1.  ])
```

Constructing Arrays

Many functions exist to create NumPy Arrays:

```
# Create an array filled with ones
>>> np.ones(5)
array([ 1.,  1.,  1.,  1.,  1.])

# Or zeros
# Specify multiple dimensions as a tuple
>>> np.zeros((2,2))
array([[ 0.,  0.],
       [ 0.,  0.]])
```

Numpy Data Types

- Numpy supports Python numerical data types as well as strings.

Numpy Data Types

- Numpy supports Python numerical data types as well as strings.
- You can also control the number of bits used to store floats and ints using the data types: np.int16, np.int32, np.int64, np.float32, np.float64, ...

Numpy Data Types

- Numpy supports Python numerical data types as well as strings.
- You can also control the number of bits used to store floats and ints using the data types: np.int16, np.int32, np.int64, np.float32, np.float64, ...

Numpy Data Types

- Numpy supports Python numerical data types as well as strings.
- You can also control the number of bits used to store floats and ints using the data types: np.int16, np.int32, np.int64, np.float32, np.float64, ...

```
# Create an array filled with ones
>>> np.array([1,2,3], dtype=np.int32)
array([1, 2, 3], dtype=int32)

>>> np.array([1,2,3], dtype=np.float64)
array([ 1.,  2.,  3.])

>>> np.array([1,0,1], dtype=bool)
array([ True, False,  True], dtype=bool)
```

Interactive NumPy Demo

Interactive Demo

1. What do the attributes `size`, `shape`, `ndim`, and `dtype` store?
2. NumPy arrays can also be created using the following three methods. What does each one do?
   - `eye`
   - `diag`
   - `empty`
3. What are the default number of bits for numpy floats and ints?

Interactive NumPy Demo

1. What do the attributes `size`, `shape`, `ndim`, and `dtype` store?

Interactive NumPy Demo

1. What do the attributes `size`, `shape`, `ndim`, and `dtype` store?
    - `size`: The number of items in the array

Interactive NumPy Demo

1. What do the attributes `size`, `shape`, `ndim`, and `dtype` store?
   - `size`: The number of items in the array
   - `shape`: The dimensions of the array in a tuple

Interactive NumPy Demo

1. What do the attributes `size`, `shape`, `ndim`, and `dtype` store?
   - `size`: The number of items in the array
   - `shape`: The dimensions of the array in a tuple
   - `ndim`: The number of dimensions

Interactive NumPy Demo

1. What do the attributes `size`, `shape`, `ndim`, and `dtype` store?
    - `size`: The number of items in the array
    - `shape`: The dimensions of the array in a tuple
    - `ndim`: The number of dimensions
    - `dtype`: The type of the contents of the array

Interactive NumPy Demo

1. What do the attributes `size`, `shape`, `ndim`, and `dtype` store?
    - `size`: The number of items in the array
    - `shape`: The dimensions of the array in a tuple
    - `ndim`: The number of dimensions
    - `dtype`: The type of the contents of the array
2. NumPy arrays can also be created using the following three methods. What does each one do?

Interactive NumPy Demo

1. What do the attributes `size`, `shape`, `ndim`, and `dtype` store?
   - `size`: The number of items in the array
   - `shape`: The dimensions of the array in a tuple
   - `ndim`: The number of dimensions
   - `dtype`: The type of the contents of the array
2. NumPy arrays can also be created using the following three methods. What does each one do?
   - `eye`: Identity

Interactive NumPy Demo

1. What do the attributes `size`, `shape`, `ndim`, and `dtype` store?
    - `size`: The number of items in the array
    - `shape`: The dimensions of the array in a tuple
    - `ndim`: The number of dimensions
    - `dtype`: The type of the contents of the array
2. NumPy arrays can also be created using the following three methods. What does each one do?
    - `eye`: Identity
    - `diag`: If the input is 1-D, returns a diagonal matrix with diagonal equal to the input. If the input is 2-D, returns the diagonal of the input.

Interactive NumPy Demo

1. What do the attributes `size`, `shape`, `ndim`, and `dtype` store?
   - `size`: The number of items in the array
   - `shape`: The dimensions of the array in a tuple
   - `ndim`: The number of dimensions
   - `dtype`: The type of the contents of the array
2. NumPy arrays can also be created using the following three methods. What does each one do?
   - `eye`: Identity
   - `diag`: If the input is 1-D, returns a diagonal matrix with diagonal equal to the input. If the input is 2-D, returns the diagonal of the input.
   - `empty`: Returns an uninitialized array. Use with caution.

Interactive NumPy Demo

1. What do the attributes `size`, `shape`, `ndim`, and `dtype` store?
   - `size`: The number of items in the array
   - `shape`: The dimensions of the array in a tuple
   - `ndim`: The number of dimensions
   - `dtype`: The type of the contents of the array
2. NumPy arrays can also be created using the following three methods. What does each one do?
   - `eye`: Identity
   - `diag`: If the input is 1-D, returns a diagonal matrix with diagonal equal to the input. If the input is 2-D, returns the diagonal of the input.
   - `empty`: Returns an uninitialized array. Use with caution.
3. What are the default number of bits for numpy floats and ints?

Interactive NumPy Demo

1. What do the attributes `size`, `shape`, `ndim`, and `dtype` store?
   - `size`: The number of items in the array
   - `shape`: The dimensions of the array in a tuple
   - `ndim`: The number of dimensions
   - `dtype`: The type of the contents of the array
2. NumPy arrays can also be created using the following three methods. What does each one do?
   - `eye`: Identity
   - `diag`: If the input is 1-D, returns a diagonal matrix with diagonal equal to the input. If the input is 2-D, returns the diagonal of the input.
   - `empty`: Returns an uninitialized array. Use with caution.
3. What are the default number of bits for numpy floats and ints?
   - `float64` and `int64`

Outline

1 Numpy and Arrays

2 Element-wise Operations and Indexing

Mathematical Operators

NumPy supports standard arithmetic operations between scalars and arrays:

Mathematical Operators

NumPy supports standard arithmetic operations between scalars and arrays:

```
>>> np.ones(3) + 2
array([ 3.,  3.,  3.])

>>> np.ones(3) - 2
array([-1., -1., -1.])

>>> np.ones(3) * 2
array([ 2.,  2.,  2.])

>>> np.ones(3) / 2
array([ 0.5,  0.5,  0.5])
```

Mathematical Operators

It also supports arithmetic operations between arrays which work on an element-wise basis:

Mathematical Operators

It also supports arithmetic operations between arrays which work on an element-wise basis:

```
>>> np.arange(1,4) + np.arange(1,4)
array([2, 4, 6])

>>> np.arange(1,4) - np.arange(1,4)
array([0, 0, 0])

# Not matrix multiplication!
>>> np.arange(1,4) * np.arange(1,4)
array([1, 4, 9])

>>> np.arange(1,4) / np.arange(1,4)
array([ 1.,  1.,  1.])
```

Mathematical Operators

It also supports arithmetic operations between arrays which work on an element-wise basis:

```
>>> np.arange(1,4) + np.arange(1,4)
array([2, 4, 6])

>>> np.arange(1,4) - np.arange(1,4)
array([0, 0, 0])

# Not matrix multiplication!
>>> np.arange(1,4) * np.arange(1,4)
array([1, 4, 9])

>>> np.arange(1,4) / np.arange(1,4)
array([ 1.,  1.,  1.])
```

Note: Python has a special function for matrix multiplication.

Logical Operators

Logical operators work in a similar fashion:

Logical Operators

Logical operators work in a similar fashion:

```
>>> np.arange(4) == 2
array([False, False,  True, False], dtype=bool)

>>> (np.array([[0,1],[2,3]]) % 2) == 0
array([[ True, False],
       [ True, False]], dtype=bool)

>>> np.array([1,2,3,4]) < np.array([0,9,9,0])
array([False,  True,  True, False], dtype=bool)
```

Indexing

Access a single element by providing an index for each dimension separated by commas:

Indexing

Access a single element by providing an index for each dimension separated by commas:

```
>>> a = np.arange(6)
>>> a
array([0, 1, 2, 3, 4, 5])
>>> a[5]
5

>>> b = np.diag(np.arange(3))
>>> b
array([[0, 0, 0],
       [0, 1, 0],
       [0, 0, 2]])
>>> b[2,2]
2
```

Slicing

Access multiple entries along a dimension using ":" notation. This is called *slicing*.

Slicing

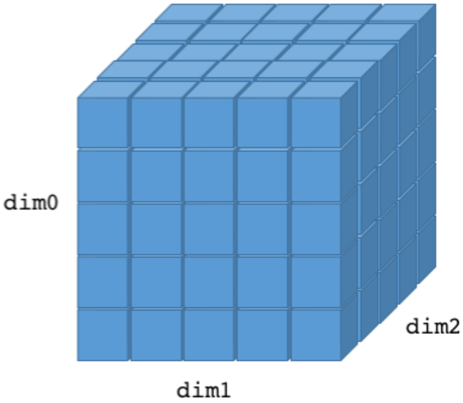
Access multiple entries along a dimension using ":" notation. This is called *slicing*.

```
>>> a = np.arange(10)
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> a[1:9:2] # start, end, step size
array([1, 3, 5, 7])

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

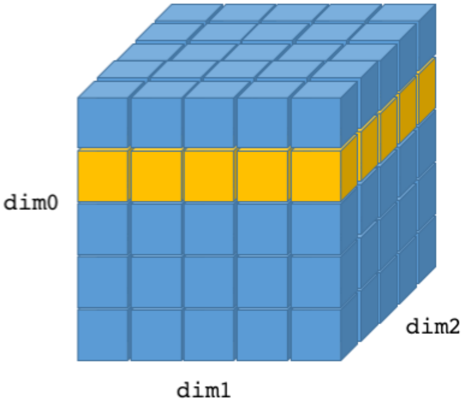
N-D Slicing

```
X = np.ones((5,5,5))
```



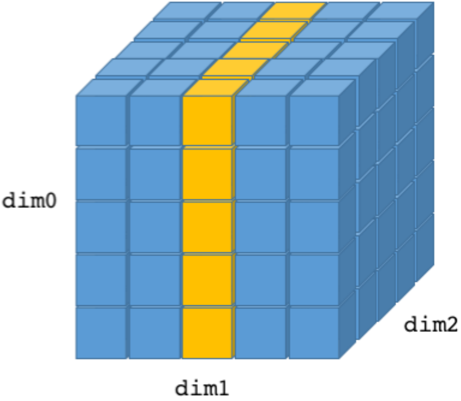
N-D Slicing

```
X = np.ones((5,5,5))
A = X[1,:,:]
```



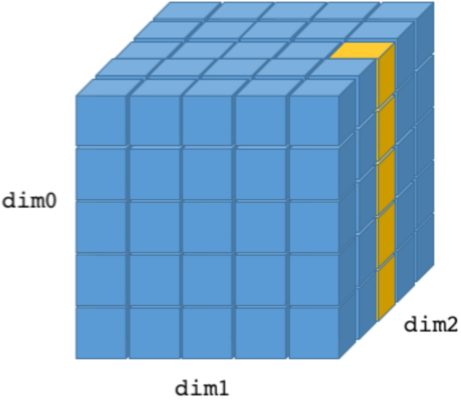
N-D Slicing

```
X = np.ones((5,5,5))
A = X[1,:,:]
B = X[:,2,:]
```



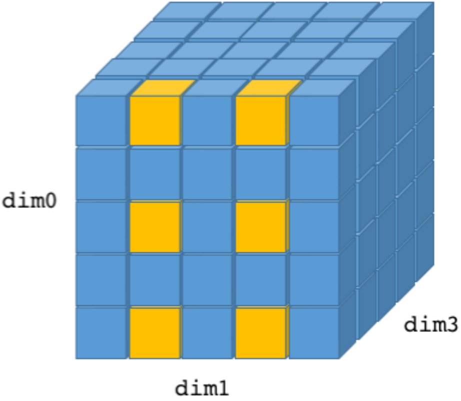
N-D Slicing

```
X = np.ones((5,5,5))
A = X[1,:,:]
B = X[:,2,:]
C = X[:,4,2]
```



N-D Slicing

```
X = np.ones((5,5,5))
A = X[1,:,:]
B = X[:,2,:]
C = X[:,4,2]
D = X[::2,1::2,0]
```



N-D Slicing

If a single index is passed for a dimension, the result will have one fewer dimensions than the original array.

```
>>> a = np.ones((2,3,4))
>>> a[:,1,:].shape
(2,4)
```

Entry/Slice Assignment

As with lists, we can assign to individual array entries or slices.

```
>>> a = np.ones((3,3))
>>> a[2,1] = 9
>>> a
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 1.,  9.,  1.]])
>>> a[:,0] = 5
>>> a
array([[ 5.,  1.,  1.],
       [ 5.,  1.,  1.],
       [ 5.,  9.,  1.]])
>>> a[0,:] = np.arange(3)
>>> a
array([[ 0.,  1.,  2.],
       [ 5.,  1.,  1.],
       [ 5.,  9.,  1.]])
```

Interactive Demo

Interactive Demo

1. Create an the following array:
   - $a_j = 2^{3j} - j,\ j = 1, ..., 10$
2. `np.all` and `np.any` are NumPy functions that operate on boolean arrays. What do these functions do?
3. How do you extract matrix B from matrix A with only indexing?

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 & 25 \end{bmatrix} \quad B = \begin{bmatrix} 25 & 23 & 21 \\ 15 & 13 & 11 \\ 5 & 3 & 1 \end{bmatrix}$$

Interactive Demo

Interactive Demo

1. Create an the following array:
   - $a_j = 2^{3j} - j,\ j = 1, ..., 10$
   - `a = 2**(3*np.arange(1,11))-np.arange(1,11)`
2. `np.all` and `np.any` are NumPy functions that operate on boolean arrays. What do these functions do?
   - `np.all` returns `True` if all elements of the array are true.
   - `np.any` returns `True` if any elements of the array are true.
3. How do you extract matrix B from matrix A with only indexing?
   - `B = A[::-2,::-2]`