NumPy Tutorial

CS-328 Numerical Methods for Visual Computing and Machine Learning

Ekrem Fatih Yilmazer & Lovro Nuic
Slides by: Sami Arpa

September 18, 2025

Introduction

About NumPy - Motivation

NumPy is the core library for scientific computing in Python.

Why NumPy?

About NumPy - Motivation

NumPy is the core library for scientific computing in Python.

Why NumPy?

- High-performance multidimensional array - **matrix operations**.

About NumPy - Motivation

NumPy is the core library for scientific computing in Python.

Why NumPy?

- High-performance multidimensional array - **matrix operations**.
- Access to a wide ecosystem of efficient numerical libraries that take NumPy arrays as input.

About NumPy - Motivation

NumPy is the core library for scientific computing in Python.

Why NumPy?

- High-performance multidimensional array - **matrix operations**.
- Access to a wide ecosystem of efficient numerical libraries that take NumPy arrays as input.
- NumPy arrays are more compact than Python lists - **efficient storage**.

Working with NumPy

To be able to use **NumPy**, we need to import it.

Working with NumPy

To be able to use **NumPy**, we need to import it.

Instead of importing the library into the namespace of our program (which would lead to clutter), we import it into a sub-namespace ("np" is a popular choice).

Working with NumPy

To be able to use **NumPy**, we need to import it.

Instead of importing the library into the namespace of our program (which would lead to clutter), we import it into a sub-namespace ("np" is a popular choice).

```
>>> import numpy as np
>>> np.pi
3.141592653589793
```

NumPy Basics

Arrays

Array basics:

Arrays

Array basics:

- A grid of values, all of the same type.

Arrays

Array basics:

- A grid of values, all of the same type.
- Indexed by a tuple of nonnegative integers.

Arrays

Array basics:

- A grid of values, all of the same type.
- Indexed by a tuple of nonnegative integers.
- The rank of the array is the number of dimensions.

Array Initialization

We can initialize NumPy arrays from Python lists.

Array Initialization

We can initialize NumPy arrays from Python lists.

```
>>> a = np.array([1, 2, 3])   # Create a rank 1 array
```

Array Initialization

We can initialize NumPy arrays from Python lists.

```
>>> a = np.array([1, 2, 3])   # Create a rank 1 array
```

```
>>> print(type(a))
<type 'numpy.ndarray'>
```

Array Initialization

We can initialize NumPy arrays from Python lists.

```
>>> a = np.array([1, 2, 3])   # Create a rank 1 array
```

```
>>> print(type(a))
<type 'numpy.ndarray'>
```

```
>>> print(a.shape)
(3,)
```

Array Initialization

We can initialize NumPy arrays from Python lists.

```
>>> a = np.array([1, 2, 3])   # Create a rank 1 array
```

```
>>> print(type(a))
<type 'numpy.ndarray'>
```

```
>>> print(a.shape)
(3,)
```

```
>>> print(a[0], a[1], a[2])
(1, 2, 3)
```

Array Initialization

We can initialize NumPy arrays from Python lists.

```
>>> a = np.array([1, 2, 3])   # Create a rank 1 array
```

```
>>> print(type(a))
<type 'numpy.ndarray'>
```

```
>>> print(a.shape)
(3,)
```

```
>>> print(a[0], a[1], a[2])
(1, 2, 3)
```

```
>>> a[0] = 5                  # Change an element of the array
>>> print(a)
[5 2 3]
```

2D Arrays

We can initialize rank 2 arrays (i.e. matrices) from *nested* Python lists.

2D Arrays

We can initialize rank 2 arrays (i.e. matrices) from *nested* Python lists.

```
>>> b = np.array([[1,2,3], [4,5,6]])  # Create a matrix
>>> print(b.shape)
(2, 3)
```

2D Arrays

We can initialize rank 2 arrays (i.e. matrices) from *nested* Python lists.

```
>>> b = np.array([[1,2,3], [4,5,6]])  # Create a matrix
>>> print(b.shape)
(2, 3)
```

```
>>> print(b[0, 0], b[0, 1], b[1, 0])
(1, 2, 4)
```

Array creation

Numpy provides many functions to create arrays:

Array creation

Numpy provides many functions to create arrays:

```
>>> a = np.zeros((2,2))   # Create an array of all zeros
>>> print(a)
[[ 0.  0.]
 [ 0.  0.]]
```

Note that writing $np.zeros(2, 2)$ won't work because it expects a tuple argument.

Array creation

Numpy provides many functions to create arrays:

```
>>> a = np.zeros((2,2))   # Create an array of all zeros
>>> print(a)
[[ 0.  0.]
 [ 0.  0.]]
```

Note that writing *np.zeros*$(2, 2)$ won't work because it expects a tuple argument.

```
>>> b = np.ones((1,2))    # Create an array of all ones
>>> print(b)
[[ 1.  1.]]
```

Array creation

```
>>> d = np.eye(2)          # Create a 2x2 identity matrix
>>> print(d)
[[ 1.  0.]
 [ 0.  1.]]
```

Array creation

```
>>> d = np.eye(2)          # Create a 2x2 identity matrix
>>> print(d)
[[ 1.  0.]
 [ 0.  1.]]
```

```
>>> e = np.random.random((2,2))  # Create a random matrix
>>> print(e)
[[ 0.42554464  0.98280237]
 [ 0.86149523  0.22512398]]
```

Array creation

```
>>> d = np.eye(2)          # Create a 2x2 identity matrix
>>> print(d)
[[ 1.  0.]
 [ 0.  1.]]
```

```
>>> e = np.random.random((2,2))  # Create a random matrix
>>> print(e)
[[ 0.42554464  0.98280237]
 [ 0.86149523  0.22512398]]
```

```
>>> f = np.linspace(1., 4., 6)
>>> print(f)
[ 1.   1.6  2.2  2.8  3.4  4. ]
```

Array indexing, slicing

Different types of indexing are possible. First option **slicing**, which is similar to Python slices.

Array indexing, slicing

Different types of indexing are possible. First option **slicing**, which is similar to Python slices.

```
>>> a = np.array([[0,1,2,3], [4,5,6,7], [8,9,10,11]])
>>> print(a)
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
>>> b = a[:2, 1:3]
>>> print(b)
[[1 2]
 [5 6]]
```

Array indexing, slicing

Danger: A slice of an array references to the same data. Changing it will modify original array.

Array indexing, slicing

Danger: A slice of an array references to the same data. Changing it will modify original array.

```
>>> a = np.array([[0,1,2,3], [4,5,6,7], [8,9,10,11]])
>>> b = a[:2, 1:3]
>>> print(a[0, 1])
1
>>> b[0, 0] = 19     # b[0, 0] is the same as a[0, 1]
>>> print(a[0, 1])
19
```

Array indexing, slicing

Danger: A slice of an array references to the same data. Changing it will modify original array.

```
>>> a = np.array([[0,1,2,3], [4,5,6,7], [8,9,10,11]])
>>> b = a[:2, 1:3]
>>> print(a[0, 1])
1
>>> b[0, 0] = 19     # b[0, 0] is the same as a[0, 1]
>>> print(a[0, 1])
19
```

```
>>> x = np.array([1, 2, 3])
>>> y = x
>>> z = np.copy(x)
>>> x[0] = 10
>>> x == y
True
>>> x == z
False
```

Array indexing, slicing

```
>>> r1 = a[1, :]   # Rank-1 view of the second row of matrix
>>> r2 = a[1:2, :] # Rank-2 view of the second row of matrix
>>> print(r1)
[4 5 6 7]
>>> print(r2)
[[4 5 6 7]]
```

Array indexing, slicing

```
>>> r1 = a[1, :]   # Rank-1 view of the second row of matrix
>>> r2 = a[1:2, :] # Rank-2 view of the second row of matrix
>>> print(r1)
[4 5 6 7]
>>> print(r2)
[[4 5 6 7]]
```

Remember that the last index in the slice notation is exclusive, so both statements select a single row.

Array indexing, indices

Indices can be used to construct arbitrary arrays from the original array.

Array indexing, indices

Indices can be used to construct arbitrary arrays from the original array.

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

Array indexing, indices

A set of indices can be used to modify arbitrary parts of an array:

Array indexing, indices

A set of indices can be used to modify arbitrary parts of an array:

```
>>> a = np.array([[1,2,3],[4,5,6]])
>>> a[[0, 1, 1], [0, 0, 1]] -= 20
>>> print(a)
[[-19   2   3]
 [-16 -15   6]]
```

Array indexing, boolean

A part of array can be selected with a given condition:

Array indexing, boolean

A part of array can be selected with a given condition:

```
>>> a = np.array([[1,2,3],[4,5,6]])
>>> print(a>3)
[[False False False]
 [ True  True  True]]
>>> print(a[a>3])
[4 5 6]
```

Datatypes

Datatypes

NumPy automatically chooses a suitable datatype (called `dtype`) for your array.

Datatypes

NumPy automatically chooses a suitable datatype (called `dtype`) for your array.

```
>>> a = np.array([0, 1, 2])   # No dtype declaration
>>> print(a.dtype)
int64
>>> a = np.array([0, 1, 2.3]) # No dtype declaration
>>> print(a.dtype)
float64
```

Datatypes

NumPy automatically chooses a suitable datatype (called `dtype`) for your array.

```
>>> a = np.array([0, 1, 2])   # No dtype declaration
>>> print(a.dtype)
int64
>>> a = np.array([0, 1, 2.3]) # No dtype declaration
>>> print(a.dtype)
float64
```

A desired datatype can be specified as well.

Datatypes

NumPy automatically chooses a suitable datatype (called `dtype`) for your array.

```
>>> a = np.array([0, 1, 2])   # No dtype declaration
>>> print(a.dtype)
int64
>>> a = np.array([0, 1, 2.3]) # No dtype declaration
>>> print(a.dtype)
float64
```

A desired datatype can be specified as well.

```
>>> a = np.array([0, 1, 2], dtype=np.int32)
>>> print(a.dtype)
int32
```

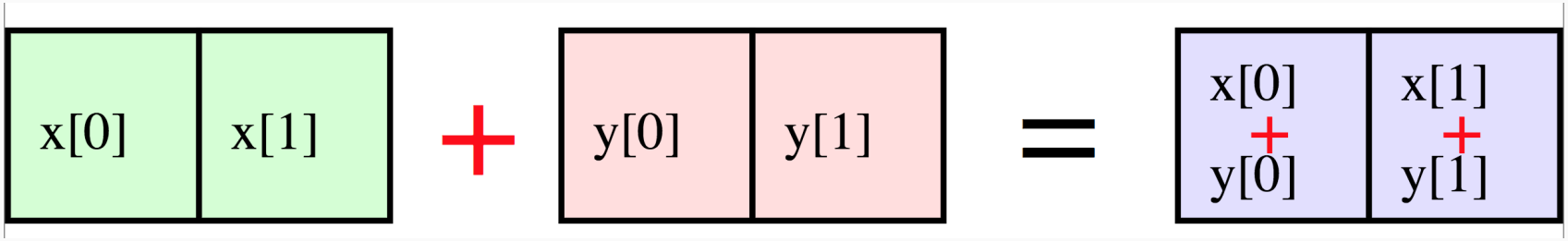
Arithmetic Operations

Universal functions

A universal function is a function that operates on **ndarrays** in an element-by-element fashion.

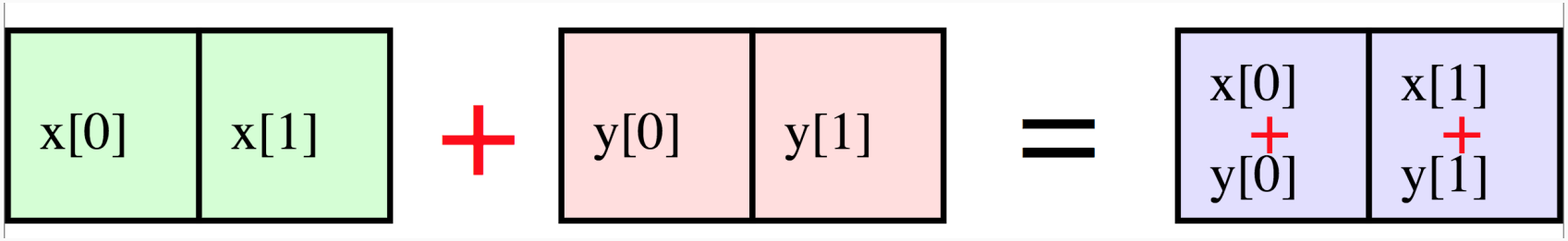
Universal functions

A universal function is a function that operates on **ndarrays** in an element-by-element fashion.



Universal functions

A universal function is a function that operates on **ndarrays** in an element-by-element fashion.



Almost all functions in NumPy are universal.

Universal functions (Elementwise operations)

Basic arithmetic operations work elementwise on arrays.

Universal functions (Elementwise operations)

Basic arithmetic operations work elementwise on arrays.

```
>>> a = np.array([2, 1], dtype=np.float64)
>>> b = np.array([4, 3], dtype=np.float64)
>>> print(a+b) # add array components element-wise
[ 6.  4.]
>>> print(a*b) # multiply array components element-wise
[ 8.  3.]
>>> print(np.sqrt(b))
[ 2.          1.73205081]
```

Other operations

Matrix-matrix/vector multiplication and dot products are expressed using the function **np.dot** (or the shorter @ operator).

Other operations

Matrix-matrix/vector multiplication and dot products are expressed using the function **np.dot** (or the shorter @ operator).

It automatically determines which of the three cases is needed based on the shape of the input arguments.

Other operations

Matrix-matrix/vector multiplication and dot products are expressed using the function **np.dot** (or the shorter @ operator).

It automatically determines which of the three cases is needed based on the shape of the input arguments.

Note that the operator $*$ performs an element-wise array multiplication.

Other operations

Matrix-matrix/vector multiplication and dot products are expressed using the function **np.dot** (or the shorter @ operator).

It automatically determines which of the three cases is needed based on the shape of the input arguments.

Note that the operator $*$ performs an element-wise array multiplication.

```
>>> a = np.array([[2, 1, 3], [3, 4, 5]], dtype=np.float64)
>>> b = np.array([[8, 2], [5, 5]], dtype=np.float64)
>>> print(b @ a) # matrix multiplication
[[ 22.  16.  34.]
 [ 25.  25.  40.]]
```

Other operations

Many other operations like `sum`, `transpose`, and `max` are available.

Other operations

Many other operations like `sum`, `transpose`, and `max` are available.

```
>>> a = np.array([[2, 1, 3], [3, 4, 5]])
>>> print(np.sum(a, axis=0))  # Compute sum of each column
[5 5 8]
```

Other operations

Many other operations like `sum`, `transpose`, and `max` are available.

```
>>> a = np.array([[2, 1, 3], [3, 4, 5]])
>>> print(np.sum(a, axis=0))  # Compute sum of each column
[5 5 8]
```

```
>>> print(a.T)
[[2 3]
 [1 4]
 [3 5]]
```

Broadcasting

Broadcasting

Broadcasting enables operations that combine arrays of different shapes. For example, we can add a constant vector to each row of a matrix.

Broadcasting

Broadcasting enables operations that combine arrays of different shapes. For example, we can add a constant vector to each row of a matrix.

```
>>> a = np.array([[0,1,2,3],[4,5,6,7],[8,9,10,11]]);
>>> b = np.array([-2,-3,0,-1])
>>> c = a + b  # Add b to each row of a
>>> print(a)
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
>>> print(c)
[[-2 -2  2  2]
 [ 2  2  6  6]
 [ 6  6 10 10]]
```

Broadcasting

Broadcasting enables operations that combine arrays of different shapes. For example, we can add a constant vector to each row of a matrix.

```
>>> a = np.array([[0,1,2,3],[4,5,6,7],[8,9,10,11]]);
>>> b = np.array([-2,-3,0,-1])
>>> c = a + b  # Add b to each row of a
>>> print(a)
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
>>> print(c)
[[-2 -2  2  2]
 [ 2  2  6  6]
 [ 6  6 10 10]]
```

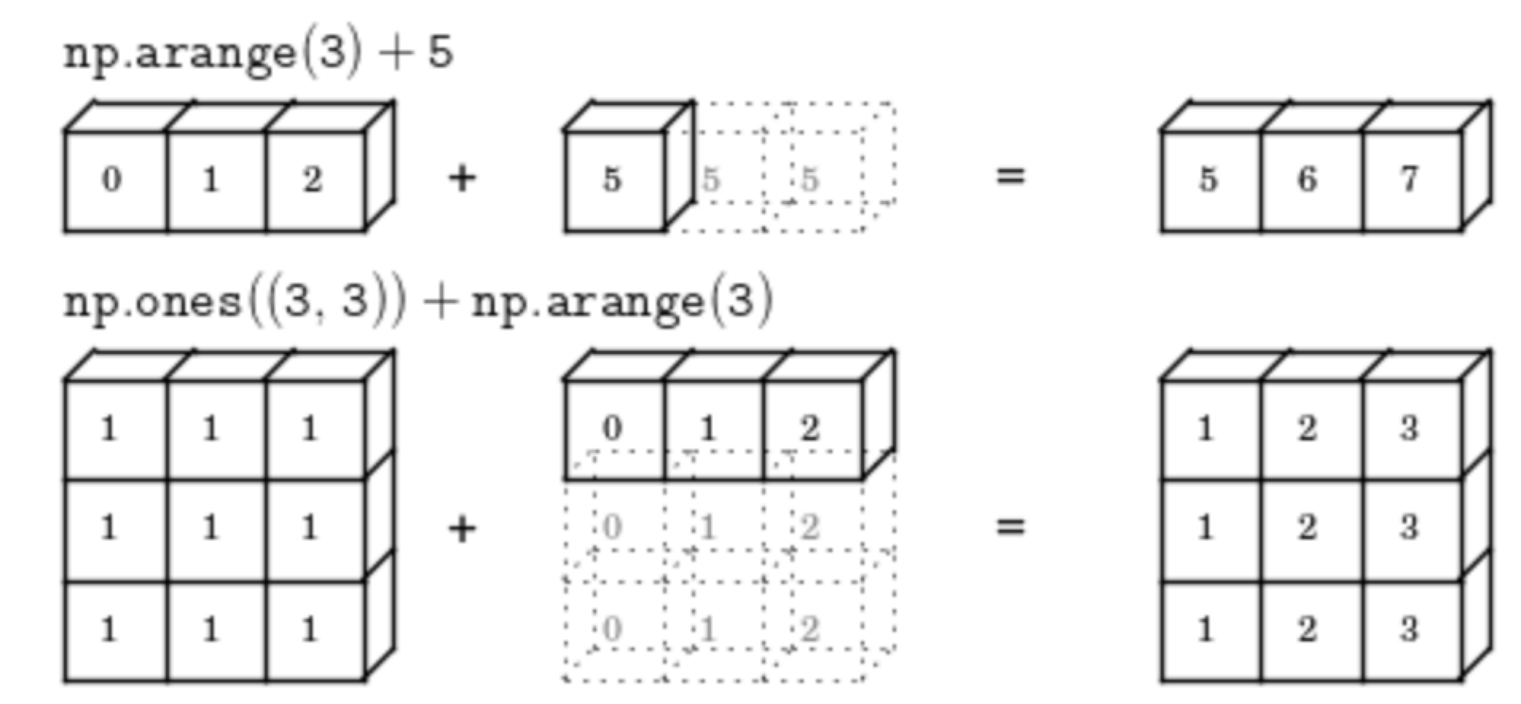
Although vector b has a different shape, NumPy assumes that b has also size $4x3$, where each row is a copy of b.

Broadcasting

A two dimensional array multiplied by a one dimensional array results in broadcasting if the number of 1D array elements matches the number of 2D array columns.

Broadcasting

A two dimensional array multiplied by a one dimensional array results in broadcasting if the number of 1D array elements matches the number of 2D array columns.

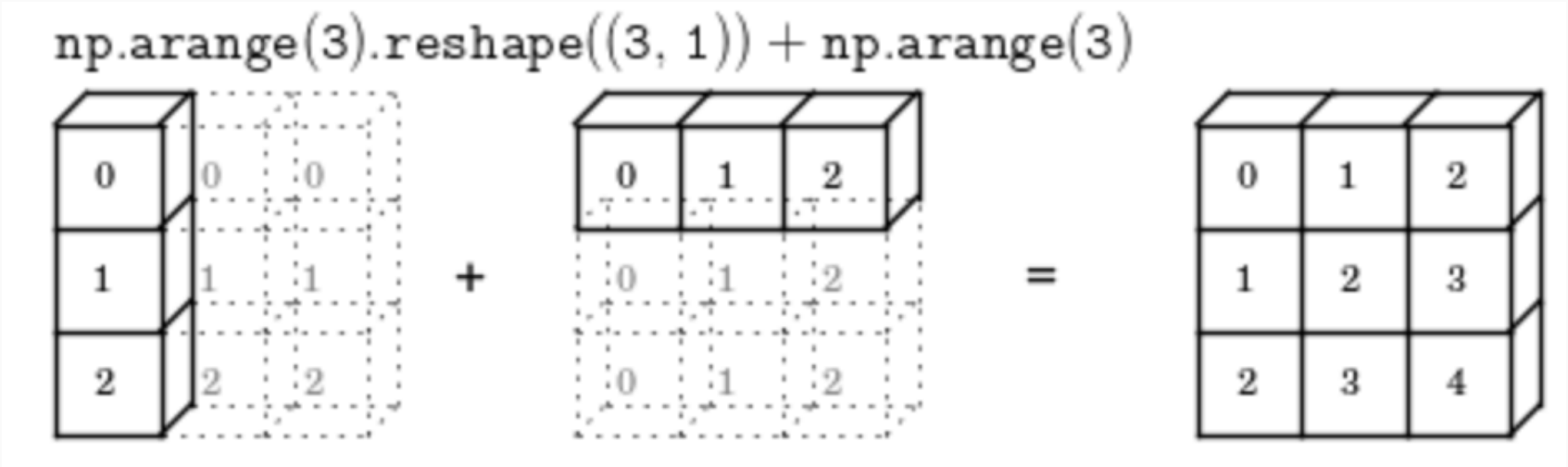


Broadcasting

In some cases, broadcasting stretches both arrays to form an output array larger than either of the initial arrays.

Broadcasting

In some cases, broadcasting stretches both arrays to form an output array larger than either of the initial arrays.



Broadcasting

When operating on two arrays, NumPy compares their shapes element-wise. It starts with the trailing dimensions, and works its way forward. Two dimensions are compatible when

Broadcasting

When operating on two arrays, NumPy compares their shapes element-wise. It starts with the trailing dimensions, and works its way forward. Two dimensions are compatible when

Broadcasting

When operating on two arrays, NumPy compares their shapes element-wise. It starts with the trailing dimensions, and works its way forward. Two dimensions are compatible when

- they are equal, or

Broadcasting

When operating on two arrays, NumPy compares their shapes element-wise. It starts with the trailing dimensions, and works its way forward. Two dimensions are compatible when

- they are equal, or
- one of them is 1.

Broadcasting

When operating on two arrays, NumPy compares their shapes element-wise. It starts with the trailing dimensions, and works its way forward. Two dimensions are compatible when

- they are equal, or
- one of them is 1.

```
A      (4d array):  8 x 1 x 6 x 1
B      (3d array):      7 x 1 x 5
Result (4d array):  8 x 7 x 6 x 5
```

Broadcasting

When operating on two arrays, NumPy compares their shapes element-wise. It starts with the trailing dimensions, and works its way forward. Two dimensions are compatible when

- they are equal, or
- one of them is 1.

```
A      (4d array):  8 x 1 x 6 x 1
B      (3d array):      7 x 1 x 5
Result (4d array):  8 x 7 x 6 x 5
```

```
A      (2d array):  5 x 4
B      (1d array):      1
Result (2d array):  5 x 4
```

Broadcasting

When operating on two arrays, NumPy compares their shapes element-wise. It starts with the trailing dimensions, and works its way forward. Two dimensions are compatible when

- they are equal, or
- one of them is 1.

```
A      (4d array):  8 x 1 x 6 x 1
B      (3d array):      7 x 1 x 5
Result (4d array):  8 x 7 x 6 x 5
```

```
A      (2d array):  5 x 4
B      (1d array):      1
Result (2d array):  5 x 4
```

```
A      (2d array):  5 x 4
B      (1d array):      4
Result (2d array):  5 x 4
```

Broadcasting

When operating on two arrays, NumPy compares their shapes element-wise. It starts with the trailing dimensions, and works its way forward. Two dimensions are compatible when

- they are equal, or
- one of them is 1.

```
A      (1d array):  3
B      (1d array):  4 # trailing dimensions do not match

A      (2d array):      2 x 1
B      (3d array):  8 x 4 x 3 # second from last dimensions mismatched
```

References

- Documentation
  https://numpy.org/doc/stable/
- Official NumPy tutorial:
  https://numpy.org/doc/stable/user/quickstart.html
- NumPy tutorial for MATLAB users:
  https://docs.scipy.org/doc/numpy/user/numpy-for-matlab-users.html
- Jupyter Notebook documentation:
  https://jupyter.readthedocs.io/en/latest/
- Others:
  http://cs231n.github.io/python-numpy-tutorial/#numpy
  http://scipy.github.io/old-wiki/pages/EricsBroadcastingDoc