

Operations on Numpy Arrays

The learning objectives of this section are:

- Manipulate arrays
 - Reshape arrays
 - Stack arrays
- Perform operations on arrays
 - Perform basic mathematical operations
 - Apply built-in functions
 - Apply your own functions
 - Apply basic linear algebra operations

Manipulating Arrays

Let's look at some ways to manipulate arrays, i.e. changing the shape, combining and splitting arrays, etc.

Reshaping Arrays

Reshaping is done using the `reshape()` function.

```
In [1]: import numpy as np

# Reshape a 1-D array to a 3 x 4 array
some_array = np.arange(0, 12).reshape(3, 4)
print(some_array)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

```
In [2]: # Can reshape it further
some_array.reshape(2, 6)
```

```
Out[2]: array([[ 0,  1,  2,  3,  4,  5],
               [ 6,  7,  8,  9, 10, 11]])
```

```
In [3]: # If you specify -1 as a dimension, the dimensions are automatically calculate
d
# -1 means "whatever dimension is needed"
some_array.reshape(4, -1)
```

```
Out[3]: array([[ 0,  1,  2],
               [ 3,  4,  5],
               [ 6,  7,  8],
               [ 9, 10, 11]])
```

`array.T` returns the transpose of an array.

```
In [4]: # Transposing an array  
some_array.T
```

```
Out[4]: array([[ 0,  4,  8],  
               [ 1,  5,  9],  
               [ 2,  6, 10],  
               [ 3,  7, 11]])
```

Stacking and Splitting Arrays

Stacking: `np.hstack()` and `np.vstack()`

Stacking is done using the `np.hstack()` and `np.vstack()` methods. For horizontal stacking, the number of rows should be the same, while for vertical stacking, the number of columns should be the same.

```
In [5]: # Creating two arrays  
array_1 = np.arange(12).reshape(3, 4)  
array_2 = np.arange(20).reshape(5, 4)  
  
print(array_1)  
print("\n")  
print(array_2)
```

```
[[ 0  1  2  3]  
 [ 4  5  6  7]  
 [ 8  9 10 11]]
```

```
[[ 0  1  2  3]  
 [ 4  5  6  7]  
 [ 8  9 10 11]  
 [12 13 14 15]  
 [16 17 18 19]]
```

```
In [6]: # vstack  
# Note that np.vstack(a, b) throws an error - you need to pass the arrays as a  
list  
np.vstack((array_1, array_2))
```

```
Out[6]: array([[ 0,  1,  2,  3],  
               [ 4,  5,  6,  7],  
               [ 8,  9, 10, 11],  
               [ 0,  1,  2,  3],  
               [ 4,  5,  6,  7],  
               [ 8,  9, 10, 11],  
               [12, 13, 14, 15],  
               [16, 17, 18, 19]])
```

Similarly, two arrays having the same number of rows can be horizontally stacked using `np.hstack((a, b))`.

Perform Operations on Arrays

Performing mathematical operations on arrays is extremely simple. Let's see some common operations.

Basic Mathematical Operations

Numpy provides almost all the basic math functions - exp, sin, cos, log, sqrt etc. The function is applied to each element of the array.

```
In [7]: # Basic mathematical operations
a = np.arange(1, 20)
```

```
# sin, cos, exp, log
print(np.sin(a))
print(np.cos(a))
print(np.exp(a))
print(np.log(a))
```

```
[ 0.84147098  0.90929743  0.14112001 -0.7568025  -0.95892427 -0.2794155
 0.6569866  0.98935825  0.41211849 -0.54402111 -0.99999021 -0.53657292
 0.42016704  0.99060736  0.65028784 -0.28790332 -0.96139749 -0.75098725
 0.14987721]
[ 0.54030231 -0.41614684 -0.9899925  -0.65364362  0.28366219  0.96017029
 0.75390225 -0.14550003 -0.91113026 -0.83907153  0.0044257  0.84385396
 0.90744678  0.13673722 -0.75968791 -0.95765948 -0.27516334  0.66031671
 0.98870462]
[ 2.71828183e+00  7.38905610e+00  2.00855369e+01  5.45981500e+01
 1.48413159e+02  4.03428793e+02  1.09663316e+03  2.98095799e+03
 8.10308393e+03  2.20264658e+04  5.98741417e+04  1.62754791e+05
 4.42413392e+05  1.20260428e+06  3.26901737e+06  8.88611052e+06
 2.41549528e+07  6.56599691e+07  1.78482301e+08]
[ 0.          0.69314718  1.09861229  1.38629436  1.60943791  1.79175947
 1.94591015  2.07944154  2.19722458  2.30258509  2.39789527  2.48490665
 2.56494936  2.63905733  2.7080502  2.77258872  2.83321334  2.89037176
 2.94443898]
```

Apply User Defined Functions

You can also apply your own functions on arrays. For e.g. applying the function $x/(x+1)$ to each element of an array.

One way to do that is by looping through the array, which is the non-numpy way. You would rather want to write **vectorized code**.

The simplest way to do that is to vectorize the function you want, and then apply it on the array. Numpy provides the `np.vectorize()` method to vectorize functions.

Let's look at both the ways to do it.

```
In [8]: print(a)
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]
```

```
In [9]: # The non-numpy way, not recommended
a_list = [x/(x+1) for x in a]
print(a_list)
```

```
[0.5, 0.6666666666666663, 0.75, 0.8000000000000004, 0.8333333333333337, 0.8571428571428571, 0.875, 0.8888888888888884, 0.9000000000000002, 0.9090909090909091, 0.9166666666666663, 0.9230769230769231, 0.9285714285714286, 0.9333333333333335, 0.9375, 0.9411764705882352, 0.9444444444444442, 0.9473684210526315, 0.9499999999999996]
```

```
In [10]: # The numpy way: vectorize the function, then apply it
f = np.vectorize(lambda x: x/(x+1))
f(a)
```

```
Out[10]: array([ 0.5          ,  0.66666667,  0.75          ,  0.8          ,  0.83333333,
                0.85714286,  0.875          ,  0.88888889,  0.9          ,  0.90909091,
                0.91666667,  0.92307692,  0.92857143,  0.93333333,  0.9375         ,
                0.94117647,  0.94444444,  0.94736842,  0.95          ])
```

```
In [11]: # Apply function on a 2-d array: Applied to each element
b = np.linspace(1, 100, 10)
f(b)
```

```
Out[11]: array([ 0.5          ,  0.92307692,  0.95833333,  0.97142857,  0.97826087,
                0.98245614,  0.98529412,  0.98734177,  0.98888889,  0.99009901])
```

This also has the advantage that you can vectorize the function once, and then apply it as many times as needed.

Apply Basic Linear Algebra Operations

NumPy provides the `np.linalg` package to apply common linear algebra operations, such as:

- `np.linalg.inv`: Inverse of a matrix
- `np.linalg.det`: Determinant of a matrix
- `np.linalg.eig`: Eigenvalues and eigenvectors of a matrix

Also, you can multiply matrices using `np.dot(a, b)`.

```
In [12]: # np.linalg documentation  
         help(np.linalg)
```

Help on package numpy.linalg in numpy:

NAME

numpy.linalg

DESCRIPTION

Core Linear Algebra Tools

Linear algebra basics:

- norm Vector or matrix norm
- inv Inverse of a square matrix
- solve Solve a linear system of equations
- det Determinant of a square matrix
- lstsq Solve linear least-squares problem
- pinv Pseudo-inverse (Moore-Penrose) calculated using a singular

value

- matrix_power Integer power of a square matrix

Eigenvalues and decompositions:

- eig Eigenvalues and vectors of a square matrix
- eigh Eigenvalues and eigenvectors of a Hermitian matrix
- eigvals Eigenvalues of a square matrix
- eigvalsh Eigenvalues of a Hermitian matrix
- qr QR decomposition of a matrix
- svd Singular value decomposition of a matrix
- cholesky Cholesky decomposition of a matrix

Tensor operations:

- tensorsolve Solve a linear tensor equation
- tensorinv Calculate an inverse of a tensor

Exceptions:

- LinAlgError Indicates a failed linear algebra operation

PACKAGE CONTENTS

_umath_linalg
info
lapack_lite
linalg
setup

DATA

```
absolute_import = _Feature((2, 5, 0, 'alpha', 1), (3, 0, 0, 'alpha', 0...
division = _Feature((2, 2, 0, 'alpha', 2), (3, 0, 0, 'alpha', 0), 8192...
print_function = _Feature((2, 6, 0, 'alpha', 2), (3, 0, 0, 'alpha', 0)...
```

FILE

c:\users\pratika\appdata\local\programs\python\python35-32\lib\site-packages\numpy\linalg__init__.py

```
In [13]: # Creating arrays
a = np.arange(1, 10).reshape(3, 3)
b= np.arange(1, 13).reshape(3, 4)
print(a)
print(b)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

```
In [14]: # Inverse
np.linalg.inv(a)
```

```
Out[14]: array([[ -4.50359963e+15,   9.00719925e+15,  -4.50359963e+15],
                [  9.00719925e+15,  -1.80143985e+16,   9.00719925e+15],
                [ -4.50359963e+15,   9.00719925e+15,  -4.50359963e+15]])
```

```
In [15]: # Determinant
np.linalg.det(a)
```

```
Out[15]: 6.6613381477509402e-16
```

```
In [16]: # Eigenvalues and eigenvectors
np.linalg.eig(a)
```

```
Out[16]: (array([ 1.61168440e+01, -1.11684397e+00, -1.30367773e-15]),
          array([[ -0.23197069, -0.78583024,  0.40824829],
                 [ -0.52532209, -0.08675134, -0.81649658],
                 [ -0.8186735 ,  0.61232756,  0.40824829]]))
```

```
In [17]: # Multiply matrices
np.dot(a, b)
```

```
Out[17]: array([[ 38,  44,  50,  56],
                [ 83,  98, 113, 128],
                [128, 152, 176, 200]])
```