Math

Understand how arithmetic and linear algebra work in NumPy.

Chapter Goals:

- Learn how to perform math operations in NumPy
- Write code using NumPy math functions

A. Arithmetic

One of the main purposes of NumPy is to perform multi-dimensional arithmetic. Using NumPy arrays, we can apply arithmetic to each element with a single operation.

The code below shows multi-dimensional arithmetic with NumPy.

```
arr = np.array([[1, 2], [3, 4]])
# Add 1 to element values
print(repr(arr + 1))
# Subtract element values by 1.2
print(repr(arr - 1.2))
# Double element values
print(repr(arr * 2))
# Halve element values
print(repr(arr / 2))
# Integer division (half)
print(repr(arr // 2))
# Square element values
print(repr(arr**2))
# Square root element values
print(repr(arr**0.5))
```

Output

0.380s

```
array([[2, 3],
       [4, 5]])
array([[-0.2,  0.8],
       [ 1.8,  2.8]])
array([[2, 4],
       [6, 8]])
array([[0.5, 1. ],
       [1.5, 2. ]])
array([[0, 1],
```

Using NumPy arithmetic, we can easily modify large amounts of numeric data with only a few operations. For example, we could convert a dataset of Fahrenheit temperatures to their equivalent Celsius form.

The code below converts Fahrenheit to Celsius in NumPy.

```
def f2c(temps):
  return (5/9)*(temps-32)

fahrenheits = np.array([32, -4, 14, -40])
celsius = f2c(fahrenheits)
print('Celsius: {}'.format(repr(celsius)))
```

Output

0.748s

```
Celsius: array([  0., -20., -10., -40.])
```

It is important to note that performing arithmetic on NumPy arrays **does not change the original array**, and instead produces a new array that is the result of the arithmetic operation.

B. Non-linear functions

Apart from basic arithmetic operations, NumPy also allows you to use non-linear functions such as exponentials and logarithms.

The function `np.exp` (https://docs.scipy.org/doc/numpy/reference/generated/numpy.exp.html) performs a base *e* exponential on an array, while the function `np.exp2` (https://docs.scipy.org/doc/numpy/reference/generated/numpy.exp2.html) performs a base 2 exponential. Likewise, `np.log` (https://docs.scipy.org/doc/numpy/reference/generated/numpy.log.html), `np.log2` (https://docs.scipy.org/doc/numpy/reference/generated/numpy.log2.html), and `np.log10` (https://docs.scipy.org/doc/numpy/reference/generated/numpy.log10.html) all perform logarithms on an input array, using base *e*, base 2, and base 10, respectively.

The code below shows various exponentials and logarithms with NumPy. Note that `np.e` and `np.pi` represent the mathematical constants *e* and π, respectively.

```
arr = np.array([[1, 2], [3, 4]])
# Raised to power of e
print(repr(np.exp(arr)))
# Raised to power of 2
print(repr(np.exp2(arr)))

arr2 = np.array([[1, 10], [np.e, np.pi]])
# Natural logarithm
print(repr(np.log(arr2)))
# Base 10 logarithm
print(repr(np.log10(arr2)))
```

Output 0.310s

```
array([[ 2.71828183,  7.3890561 ],
       [20.08553692, 54.59815003]])
array([[ 2.,  4.],
       [ 8., 16.]])
array([[0.        , 2.30258509],
```

```
       [1.        , 1.14472989]])
array([[0.        , 1.        ],
```

To do a regular power operation with any base, we use `np.power` (https://docs.scipy.org/doc/numpy/reference/generated/numpy.power.html) . The first argument to the function is the base, while the second is the power. If the base or power is an array rather than a single number, the operation is applied to every element in the array.

The code below shows examples of using `np.power` .

```
arr = np.array([[1, 2], [3, 4]])
# Raise 3 to power of each number in arr
print(repr(np.power(3, arr)))
arr2 = np.array([[10.2, 4], [3, 5]])
# Raise arr2 to power of each number in arr
print(repr(np.power(arr2, arr)))
```

Output 0.625s

```
array([[ 3,  9],
       [27, 81]])
array([[ 10.2,  16. ],
       [ 27. , 625. ]])
```

In addition to exponentials and logarithms, NumPy has various other mathematical functions, which are listed here (https://docs.scipy.org/doc/numpy/reference/routines.math.html).

C. Matrix multiplication

Since NumPy arrays are basically vectors and matrices, it makes sense that there are functions for dot products and matrix multiplication. Specifically, the main function to use is `np.matmul` (https://docs.scipy.org/doc/numpy/reference/generated/numpy.matmul.html), which takes two vector/matrix arrays as input and produces a dot product or matrix multiplication.

The code below shows various examples of matrix multiplication. When both inputs are 1-D, the output is the dot product.

Note that the dimensions of the two input matrices must be valid for a matrix multiplication. Specifically, the second dimension of the first matrix must equal the first dimension of the second matrix, otherwise `np.matmul` will result in a `ValueError`.

```
arr1 = np.array([1, 2, 3])
arr2 = np.array([-3, 0, 10])
print(np.matmul(arr1, arr2))

arr3 = np.array([[1, 2], [3, 4], [5, 6]])
arr4 = np.array([[-1, 0, 1], [3, 2, -4]])
print(repr(np.matmul(arr3, arr4)))
print(repr(np.matmul(arr4, arr3)))
# This will result in ValueError
print(repr(np.matmul(arr3, arr3)))
```

Output 0.412s

```
27
array([[  5,   4,  -7],
       [  9,   8, -13],
       [ 13,  12, -19]])
array([[  4,   4],
       [-11, -10]])
```

```
Traceback (most recent call last):
  File "main.py", line 13, in <module>
    print(repr(np.matmul(arr3, arr3)))
ValueError: shapes (3,2) and (3,2) not aligned: 2 (dim 1) != 3 (dim 0)
```

Time to Code!

We'll create a couple of matrix arrays to perform our math operations on. The first array will represent the matrix:

-0.5	0.8	-0.1
0.0	-1.2	1.3

The second array will represent the matrix:

1.2	3.1
1.2	0.3
1.5	2.2

Set `arr` equal to `np.array` applied to a list of lists representing the first matrix.

Then set `arr2` equal to `np.array` applied to a list of lists representing the second matrix.

```
arr = np.array([[-0.5, 0.8, -0.1], [0.0, -1.2, 1.3]])
arr2 = np.array([[1.2, 3.1], [1.2, 0.3], [1.5, 2.2]])


```

Show Results | Show Console

2 of 2 Tests Passed

Result	Input	Expected Output	Actual Output	Reason
✓		array([[-0.5, 0.8, -0.1], [0. , ̄	array([[-0.5, 0.8, -0.1], [0. , ̄	Correct value for arr, good job!

Result	Input	Expected Output	Actual Output	Reason
✓		array([[1.2, 3.1], [1.2, 0.3], ⁻	array([[1.2, 3.1], [1.2, 0.3], ⁻	Correct value for arr2, good job!

0.398s

Next we'll apply some arithmetic to `arr`. Specifically, we'll do multiplication, addition, and squaring.

Set `multiplied` equal to `arr` multiplied by `np.pi`.

Then set `added` equal to the result of adding `arr` and `multiplied`.

Finally, set `squared` equal to `added` with each of its elements squared.

```
multiplied = arr * np.pi
added = arr + multiplied
squared = added**2
```

Show Results | Show Console

3 of 3 Tests Passed

Result	Input	Expected Output	Actual Output	Reason
✓		array([[-1.57079633, 2.51327412, -0.314 ⁻	array([[-1.57079633, 2.51327412, -0.314 ⁻	Correct value for multiplied, good job!
✓		array([[-2.07079633, 3.31327412, -0.414 ⁻	array([[-2.07079633, 3.31327412, -0.414 ⁻	Correct value for added, good job!
✓		array([[4.28819743, 10.97778541, 0.171 ⁻	array([[4.28819743, 10.97778541, 0.171 ⁻	Correct value for squared, good job!

0.416s

After the arithmetic operations, we'll apply the base *e* exponential and logarithm to our array matrices.

Set `exponential` equal to `np.exp` applied to `squared`.

Then set `logged` equal to `np.log` applied to `arr2`.

```
exponential = np.exp(squared)
logged = np.log(arr2)
```

Show Results | Show Console

2 of 2 Tests Passed

Result	Input	Expected Output	Actual Output	Reason
✓		array([[7.28350596e+01, 5.85587272e+04,	array([[7.28350596e+01, 5.85587272e+04,	Correct value for exponentia
✓		array([[0.18232156, 1.13140211],	array([[0.18232156, 1.13140211],	Correct value for logged, good job!

0.798s

Note that `exponential` has shape `(2, 3)` and `logged` has shape `(3, 2)`. So we can perform matrix multiplication both ways.

Set `matmul1` equal to `np.matmul` with first argument `logged` and second argument `exponential`. Note that `matmul1` will have shape `(3, 3)`.

Then set `matmul2` equal to `np.matmul` with first argument `exponential` and second argument `logged`. Note that `matmul2` will have shape `(2, 2)`.

```
matmul1 = np.matmul(logged, exponential)
matmul2 = np.matmul(exponential, logged)
```

Show Results | Show Console

2 of 2 Tests Passed

Result	Input	Expected Output	Actual Output	Reason
✓		array([[1.44108036e+01, 6.03529115e+10 …	array([[1.44108036e+01, 6.03529115e+10 …	Correct value for matmul1, good job!
✓		array([[1.06902790e+04, -7.04197733e+04 …	array([[1.06902790e+04, -7.04197733e+04 …	Correct value for matmul2, good job!

0.763s