Introduction to NumPy

MRE/EME 5983 Robot Operating Systems

Overview

- What is NumPy?

- NumPy Overview

- This overview follows “Python Machine Learning” by Wei-Meng Lee

What Is NumPy?

- An extension to the Python programming language that adds support for large, multidimensional arrays and matrices, along with a large library of high-level mathematical functions to operate on these arrays

- Why?
  - In Python, you usually use the list data type to store a collection of items
  - Unlike arrays, a Python list does not need to contain elements of the same type
    - Some list examples

```
list1 = [1,2,3,4,5]

list2 = [1,"Hello",3.14,True,5]
```

Source: Python Machine Learning by Wei-Meng Lee

Creating NumPy Arrays

- Before using NumPy, you first need to import the NumPy package

```
import numpy as np
```

- Creating arrays using NumPy functions

```
a1 = np.arange(10)          # creates a range from 0 to 9
print(a1)                   # [0 1 2 3 4 5 6 7 8 9]
print(a1.shape)             # (10,)

a2 = np.arange(0,10,2)      # creates a range from 0 to 9, step 2
print(a2)                   # [0 2 4 6 8]

a3 = np.zeros(5)            # create an array with all 0s
print(a3)                   # [ 0.  0.  0.  0.  0.]
print(a3.shape)             # (5,)
```

- The examples above create rank 1 arrays (one-dimensional)

Source: Python Machine Learning by Wei-Meng Lee

Creating NumPy Arrays – Higher Rank Order

- We can create higher rank order arrays

```
a4 = np.zeros((2,3))       # array of rank 2 with all 0s; 2 rows and 3
                           # columns
print(a4.shape)            # (2,3)

print(a4)
'''
[[ 0.  0.  0.]
 [ 0.  0.  0.]]
'''
```

- Other examples

```
a6 = np.eye(4)
print(a6)
'''
[[ 1.  0.  0.  0.]
 [ 0.  1.  0.  0.]
 [ 0.  0.  1.  0.]
 [ 0.  0.  0.  1.]]
'''
```

```
a7 = np.random.random((2,4)) # rank 2 array (2 rows 4 columns) with
                             # random values
                             # in the half-open interval [0.0, 1.0)
print(a7)
'''
[[ 0.48255806  0.23928884  0.99861279  0.4624779 ]
 [ 0.18721584  0.71287041  0.84619432  0.65990083]]
'''
```

Source: Python Machine Learning by Wei-Meng Lee

Creating NumPy Arrays From Python Lists

- We can create NumPy arrays from Python lists

```
list1 = [1,2,3,4,5]  # list1 is a list in Python
r1 = np.array(list1) # rank 1 array
print(r1)            # [1 2 3 4 5]


list2 = [6,7,8,9,0]
r2 = np.array([list1,list2])     # rank 2 array
print(r2)
'''
[[1 2 3 4 5]
 [6 7 8 9 0]]
'''
```

Source: Python Machine Learning by Wei-Meng Lee

NumPy Arrays Indexing

- We can create NumPy arrays from Python lists

```
list1 = [1,2,3,4,5]  # list1 is a list in Python
r1 = np.array(list1) # rank 1 array
print(r1)            # [1 2 3 4 5]
print(r1[0])         # 1
print(r1[1])         # 2
```

```
list2 = [6,7,8,9,0]
r2 = np.array([list1,list2])     # rank 2 array
print(r2)
'''
[[1 2 3 4 5]
 [6 7 8 9 0]]
'''
print(r2.shape)              # (2,5) - 2 rows and 5 columns
print(r2[0,0])               # 1
print(r2[0,1])               # 2
print(r2[1,0])               # 6
```

Source: Python Machine Learning by Wei-Meng Lee

NumPy Arrays Boolean Indexing

- We can use an array of Booleans to select a subset of items from an array

```
list1 = [1,2,3,4,5]   # list1 is a list in Python
r1 = np.array(list1) # rank 1 array
print(r1)            # [1 2 3 4 5]
```

- Get the list of r1 elements greater than 2

```
print(r1>2)     # [False False  True  True  True]
```

- Print the r1 elements greater than 2

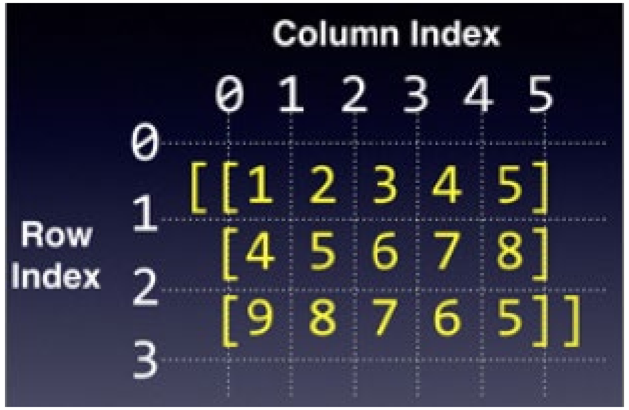
```
print(r1[r1>2])    # [3 4 5]
```

Source: Python Machine Learning by Wei-Meng Lee

NumPy Arrays Slicing

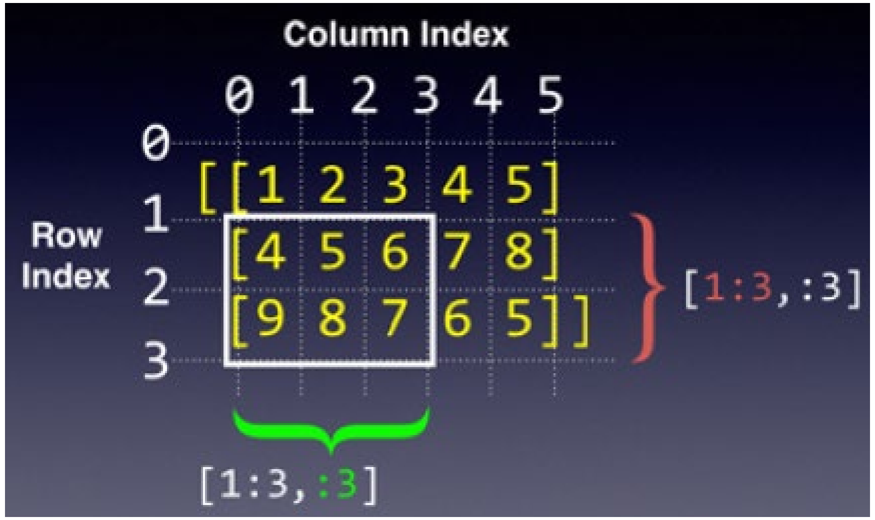
- Similar to Python lists, we can slice sections for NumPy arrays

```
a = np.array([[1,2,3,4,5],
              [4,5,6,7,8],
              [9,8,7,6,5]])    # rank 2 array
print(a)
'''
[[1 2 3 4 5]
 [4 5 6 7 8]
 [9 8 7 6 5]]
'''
```




```
b1 = a[1:3, :3]    # row 1 to 3 (not inclusive) and first 3 columns
print(b1)


[[4 5 6]
 [9 8 7]]
```



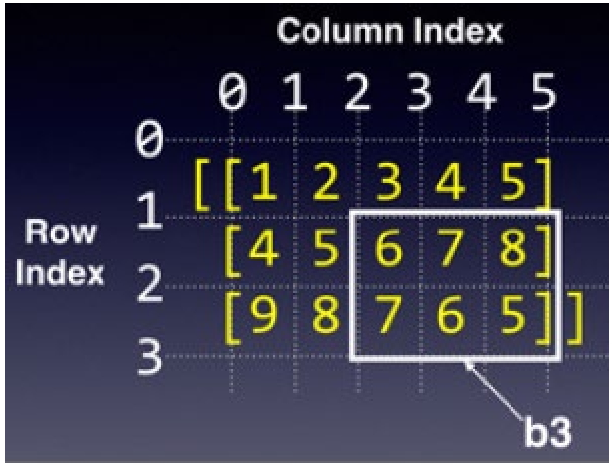
Source: Python Machine Learning by Wei-Meng Lee

NumPy Arrays Slices are References

- Similar to Python lists, we can slice sections for NumPy arrays

```
a = np.array([[1,2,3,4,5],
              [4,5,6,7,8],
              [9,8,7,6,5]])    # rank 2 array

b3 = a[1:, 2:]      # row 1 onwards and column 2 onwards
                    # b3 is now pointing to a subset of a
print(b3)
[[6 7 8]
 [7 6 5]]
```




```
b3[0,2] = 88        # b3[0,2] is pointing to a[1,4]; modifying it will
                    # modify the original array
print(a)
[[ 1  2  3  4  5]
 [ 4  5  6  7 88]
 [ 9  8  7  6  5]]
```

Source: Python Machine Learning by Wei-Meng Lee

NumPy Array Mathematics

- By default, NumPy performs element-wise array mathematics

```
x1 = np.array([[1,2,3],[4,5,6]])
y1 = np.array([[7,8,9],[2,3,4]])
print(x1 + y1)     # same as np.add(x1,y1)
[[ 8 10 12]
 [ 6  8 10]]

print(x1 - y1)     # same as np.subtract(x1,y1)
'''
[[-6 -6 -6]
 [ 2  2  2]]
'''
```

```
print(x1 * y1)     # same as np.multiply(x1,y1)
'''
[[ 7 16 27]
 [ 8 15 24]]
'''

print(x1 / y1)     # same as np.divide(x1,y1)
'''
[[ 0.14285714  0.25        0.33333333]
 [ 2.          1.66666667  1.5       ]]
'''
```

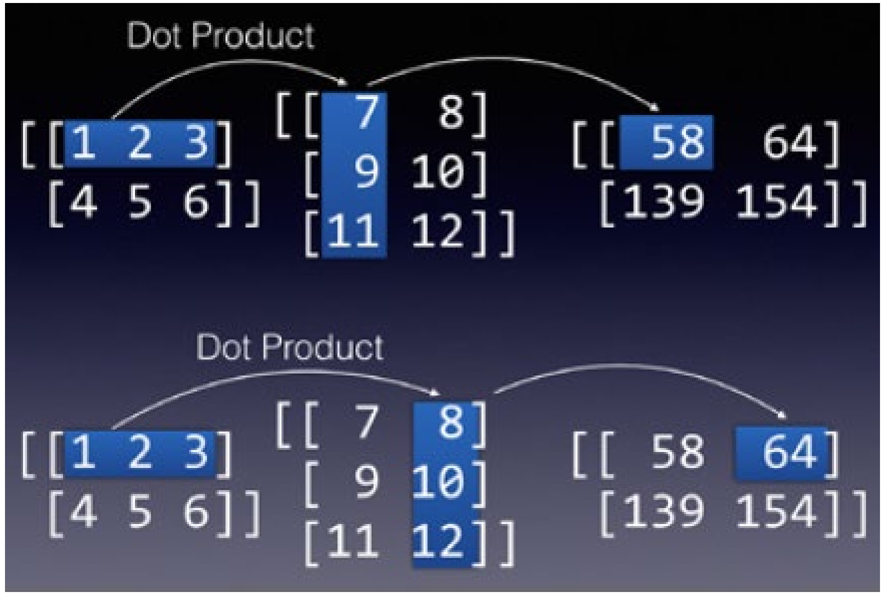
Source: Python Machine Learning by Wei-Meng Lee

NumPy Array Matrix Mathematics

- We can use dot product to perform matrix operations

```
x = np.array([2,3])
y = np.array([4,2])
np.dot(x,y)  # 2x4 + 3x2 = 14


x2 = np.array([[1,2,3],[4,5,6]])
y2 = np.array([[7,8],[9,10], [11,12]])
print(np.dot(x2,y2))                    # matrix multiplication
'''
[[ 58  64]
 [139 154]]
 '''
```



Source: Python Machine Learning by Wei-Meng Lee

NumPy **Matrices**

- NumPy also offers matrices natively…

```
x1 = np.array([[1,2],[4,5]])
y1 = np.array([[7,8],[2,3]])
print(x1 * y1)     # element-by-element multiplication
'''
[[ 7 16]
 [ 8 15]]
'''

x2 = np.matrix([[1,2],[4,5]])
y2 = np.matrix([[7,8],[2,3]])
print(x2 * y2)    # dot product; same as np.dot()
'''
[[11 14]
 [38 47]]
'''
```

Source: Python Machine Learning by Wei-Meng Lee

NumPy Array Sorting

- NumPy offers very efficient sorting algorithms

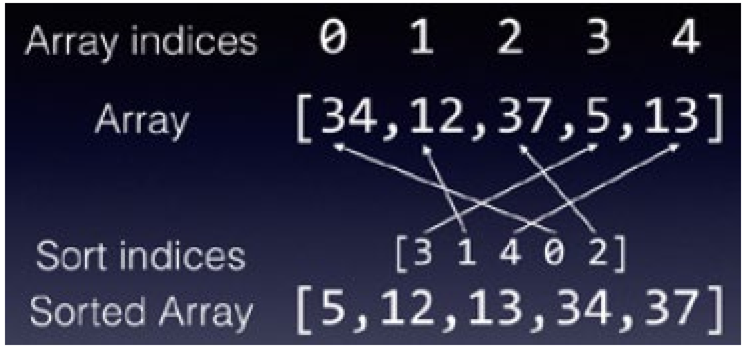
```
ages = np.array([34,12,37,5,13])
sorted_ages = np.sort(ages)    # does not modify the original array
print(sorted_ages)             # [ 5 12 13 34 37]
print(ages)                    # [34 12 37  5 13]
```

- If you would like to sort the actual array, use .sort on the array

```
ages.sort()                    # modifies the array
print(ages)                    # [ 5 12 13 34 37]
```

- Argument sort can be very useful

```
ages = np.array([34,12,37,5,13])
print(ages.argsort())          # [3 1 4 0 2]
```



Source: Python Machine Learning by Wei-Meng Lee

NumPy Array Assignment and Copy

- Be careful to watch Python variable references

```
list1 = [[1,2,3,4], [5,6,7,8]]
a1 = np.array(list1)
print(a1)
'''
[[1 2 3 4]
 [5 6 7 8]]
'''
a2 = a1    # creates a copy by reference
a2[0][0] = 11       # make some changes to a2
print(a1)           # affects a1
'''
[[11  2  3  4]
 [ 5  6  7  8]]
'''

print(a2)
'''
[[11  2  3  4]
 [ 5  6  7  8]]
'''
```

```
list1 = [[1,2,3,4], [5,6,7,8]]
a1 = np.array(list1)
a2 = a1.copy()     # create a copy of a1 by value (deep copy)

a1[0][0] = 11      # make some changes in a1
print(a1)
'''
[[11  2  3  4]
 [ 5  6  7  8]]
'''

print(a2)          # changes is not seen in a2
'''
[[1 2 3 4]
 [5 6 7 8]]
'''
```

Source: Python Machine Learning by Wei-Meng Lee

NumPy Mathematical Functions

Trigonometric functions

`sin`(x, /[, out, where, casting, order, ...])	Trigonometric sine, element-wise.
`cos`(x, /[, out, where, casting, order, ...])	Cosine element-wise.
`tan`(x, /[, out, where, casting, order, ...])	Compute tangent element-wise.
`arcsin`(x, /[, out, where, casting, order, ...])	Inverse sine, element-wise.
`arccos`(x, /[, out, where, casting, order, ...])	Trigonometric inverse cosine, element-wise.
`arctan`(x, /[, out, where, casting, order, ...])	Trigonometric inverse tangent, element-wise.
`hypot`(x1, x2, /[, out, where, casting, ...])	Given the "legs" of a right triangle, return its hypotenuse.
`arctan2`(x1, x2, /[, out, where, casting, ...])	Element-wise arc tangent of `x1/x2` choosing the quadrant correctly.
`degrees`(x, /[, out, where, casting, order, ...])	Convert angles from radians to degrees.
`radians`(x, /[, out, where, casting, order, ...])	Convert angles from degrees to radians.
`unwrap`(p[, discont, axis, period])	Unwrap by taking the complement of large deltas with respect to the period.
`deg2rad`(x, /[, out, where, casting, order, ...])	Convert angles from degrees to radians.
`rad2deg`(x, /[, out, where, casting, order, ...])	Convert angles from radians to degrees.

Hyperbolic functions

`sinh`(x, /[, out, where, casting, order, ...])	Hyperbolic sine, element-wise.
`cosh`(x, /[, out, where, casting, order, ...])	Hyperbolic cosine, element-wise.
`tanh`(x, /[, out, where, casting, order, ...])	Compute hyperbolic tangent element-wise.
`arcsinh`(x, /[, out, where, casting, order, ...])	Inverse hyperbolic sine element-wise.
`arccosh`(x, /[, out, where, casting, order, ...])	Inverse hyperbolic cosine, element-wise.
`arctanh`(x, /[, out, where, casting, order, ...])	Inverse hyperbolic tangent element-wise.

Rounding

`around`(a[, decimals, out])	Evenly round to the given number of decimals.
`round_`(a[, decimals, out])	Round an array to the given number of decimals.
`rint`(x, /[, out, where, casting, order, ...])	Round elements of the array to the nearest integer.
`fix`(x[, out])	Round to nearest integer towards zero.
`floor`(x, /[, out, where, casting, order, ...])	Return the floor of the input, element-wise.
`ceil`(x, /[, out, where, casting, order, ...])	Return the ceiling of the input, element-wise.
`trunc`(x, /[, out, where, casting, order, ...])	Return the truncated value of the input, element-wise.

NumPy Mathematical Functions

Sums, products, differences

prod(a[, axis, dtype, out, keepdims, …])	Return the product of array elements over a given axis.
sum(a[, axis, dtype, out, keepdims, …])	Sum of array elements over a given axis.
nanprod(a[, axis, dtype, out, keepdims, …])	Return the product of array elements over a given axis treating Not a Numbers (NaNs) as ones.
nansum(a[, axis, dtype, out, keepdims, …])	Return the sum of array elements over a given axis treating Not a Numbers (NaNs) as zero.
cumprod(a[, axis, dtype, out])	Return the cumulative product of elements along a given axis.
cumsum(a[, axis, dtype, out])	Return the cumulative sum of the elements along a given axis.
nancumprod(a[, axis, dtype, out])	Return the cumulative product of array elements over a given axis treating Not a Numbers (NaNs) as one.
nancumsum(a[, axis, dtype, out])	Return the cumulative sum of array elements over a given axis treating Not a Numbers (NaNs) as zero.
diff(a[, n, axis, prepend, append])	Calculate the n-th discrete difference along the given axis.
ediff1d(ary[, to_end, to_begin])	The differences between consecutive elements of an array.
gradient(f, *varargs[, axis, edge_order])	Return the gradient of an N-dimensional array.
cross(a, b[, axisa, axisb, axisc, axis])	Return the cross product of two (arrays of) vectors.
trapz(y[, x, dx, axis])	Integrate along the given axis using the composite trapezoidal rule.

Exponents and logarithms

exp(x, /[, out, where, casting, order, …])	Calculate the exponential of all elements in the input array.
expm1(x, /[, out, where, casting, order, …])	Calculate `exp(x) - 1` for all elements in the array.
exp2(x, /[, out, where, casting, order, …])	Calculate *2**p* for all *p* in the input array.
log(x, /[, out, where, casting, order, …])	Natural logarithm, element-wise.
log10(x, /[, out, where, casting, order, …])	Return the base 10 logarithm of the input array, element-wise.
log2(x, /[, out, where, casting, order, …])	Base-2 logarithm of *x*.
log1p(x, /[, out, where, casting, order, …])	Return the natural logarithm of one plus the input array, element-wise.
logaddexp(x1, x2, /[, out, where, casting, …])	Logarithm of the sum of exponentiations of the inputs.
logaddexp2(x1, x2, /[, out, where, casting, …])	Logarithm of the sum of exponentiations of the inputs in base-2.

Other special functions

i0(x)	Modified Bessel function of the first kind, order 0.
sinc(x)	Return the normalized sinc function.

NumPy Mathematical Functions

Arithmetic operations

`add`(x1, x2, /[, out, where, casting, order, ...])	Add arguments element-wise.
`reciprocal`(x, /[, out, where, casting, ...])	Return the reciprocal of the argument, element-wise.
`positive`(x, /[, out, where, casting, order, ...])	Numerical positive, element-wise.
`negative`(x, /[, out, where, casting, order, ...])	Numerical negative, element-wise.
`multiply`(x1, x2, /[, out, where, casting, ...])	Multiply arguments element-wise.
`divide`(x1, x2, /[, out, where, casting, ...])	Divide arguments element-wise.
`power`(x1, x2, /[, out, where, casting, ...])	First array elements raised to powers from second array, element-wise.
`subtract`(x1, x2, /[, out, where, casting, ...])	Subtract arguments, element-wise.
`true_divide`(x1, x2, /[, out, where, ...])	Divide arguments element-wise.
`floor_divide`(x1, x2, /[, out, where, ...])	Return the largest integer smaller or equal to the division of the inputs.
`float_power`(x1, x2, /[, out, where, ...])	First array elements raised to powers from second array, element-wise.
`fmod`(x1, x2, /[, out, where, casting, ...])	Returns the element-wise remainder of division.
`mod`(x1, x2, /[, out, where, casting, order, ...])	Returns the element-wise remainder of division.
`modf`(x[, out1, out2], / [[, out, where, ...])	Return the fractional and integral parts of an array, element-wise.
`remainder`(x1, x2, /[, out, where, casting, ...])	Returns the element-wise remainder of division.
`divmod`(x1, x2[, out1, out2], / [[, out, ...])	Return element-wise quotient and remainder simultaneously.

Handling complex numbers

`angle`(z[, deg])	Return the angle of the complex argument.
`real`(val)	Return the real part of the complex argument.
`imag`(val)	Return the imaginary part of the complex argument.
`conj`(x, /[, out, where, casting, order, ...])	Return the complex conjugate, element-wise.
`conjugate`(x, /[, out, where, casting, ...])	Return the complex conjugate, element-wise.

Extrema Finding

`maximum`(x1, x2, /[, out, where, casting, ...])	Element-wise maximum of array elements.
`fmax`(x1, x2, /[, out, where, casting, ...])	Element-wise maximum of array elements.
`amax`(a[, axis, out, keepdims, initial, where])	Return the maximum of an array or maximum along an axis.
`nanmax`(a[, axis, out, keepdims, initial, where])	Return the maximum of an array or maximum along an axis, ignoring any NaNs.
`minimum`(x1, x2, /[, out, where, casting, ...])	Element-wise minimum of array elements.
`fmin`(x1, x2, /[, out, where, casting, ...])	Element-wise minimum of array elements.
`amin`(a[, axis, out, keepdims, initial, where])	Return the minimum of an array or minimum along an axis.
`nanmin`(a[, axis, out, keepdims, initial, where])	Return minimum of an array or minimum along an axis, ignoring any NaNs.

NumPy Mathematical Functions

Miscellaneous

convolve(a, v[, mode])	Returns the discrete, linear convolution of two one-dimensional sequences.
clip(a, a_min, a_max[, out])	Clip (limit) the values in an array.
sqrt(x, /[, out, where, casting, order, ...])	Return the non-negative square-root of an array, element-wise.
cbrt(x, /[, out, where, casting, order, ...])	Return the cube-root of an array, element-wise.
square(x, /[, out, where, casting, order, ...])	Return the element-wise square of the input.
absolute(x, /[, out, where, casting, order, ...])	Calculate the absolute value element-wise.
fabs(x, /[, out, where, casting, order, ...])	Compute the absolute values element-wise.
sign(x, /[, out, where, casting, order, ...])	Returns an element-wise indication of the sign of a number.
heaviside(x1, x2, /[, out, where, casting, ...])	Compute the Heaviside step function.
nan_to_num(x[, copy, nan, posinf, neginf])	Replace NaN with zero and infinity with large finite numbers (default behaviour) or with the numbers defined by the user using the `nan`, *posinf* and/or *neginf* keywords.
real_if_close(a[, tol])	If input is complex with all imaginary parts close to zero, return real parts.
interp(x, xp, fp[, left, right, period])	One-dimensional linear interpolation for monotonically increasing sample points.

Reading and Writing .csv Files

- Reading .csv files

```
>>> arr = np.genfromtxt('data.csv', delimiter=',')
>>> arr
array([[0., 0., 0.],
       [1., 2., 3.],
       [4., 5., 6.],
       [7., 8., 9.],
       [6., 5., 4.],
       [3., 2., 1.]])
```

data.csv

```
0.0, 0.0, 0.0
1.0, 2.0, 3.0
4.0, 5.0, 6.0
7.0, 8.0, 9.0
6.0, 5.0, 4.0
3.0, 2.0, 1.0
```

- Writing .csv files

```
np.savetxt('data_out.csv', arr, delimiter=',',format='%.2f')
```

data_out.csv

```
0.00,0.00,0.00
1.00,2.00,3.00
4.00,5.00,6.00
7.00,8.00,9.00
6.00,5.00,4.00
3.00,2.00,1.00
```

Summary

- We had a brief introduction to NumPy and will leverage these concepts in upcoming lectures and assignments