Working with NumPy

NumPy library

- NumPy is one of the main Python libraries that is used for numerical computing
  - typically comes preinstalled with most Python IDEs
- It provides support for creating scalars, vectors, matrices, and tensors
- The library can be used to perform linear algebra, element-wise operations, statistical analysis, data storage etc.

- To stary using NumPy, you will have to import the library in your Python session as

```
import numpy as np
```

Scalars, vectors, matrices, and tensors

- Scalars
  - A scalar is a single digit value
  - It has no direction

```
import numpy as np
scalar = np.array( 10 )
print( scalar.shape )
```

Scalars, vectors, matrices, and tensors

- Vectors
  - A vector is one-dimensional sequence of numbers, which as a result has a magnitude and direction

```
vector = np.array( [10, 20, 30, 40, 50] )
print( vector.shape )
```

Scalars, vectors, matrices, and tensors

- Matrices
  - A matrix is a two-dimensional array of numbers
  - One can create a matrix by first defining a list of lists of numerical values that correspond to the rows and columns of the matrix

```
matrix = np.array( [ [10, 20], [30, 40] ] )
print( matrix.shape )
```

Scalars, vectors, matrices, and tensors

- Tensors
  - A tensor is a multi-dimensional array of numbers
  - Tensors can have any number of dimensions and are represented using arrays of arrays
  - Tensor is typically used for numerical array of 3-dimensions and more.
  - Similar to matrix, a create a tensor by first defining a list of lists of numerical values that correspond to the rows and columns of the matrix

```
tensor = np.array([[[10, 20, 30], [40, 50, 60]], [[100, 200, 300], [400, 500, 600]]])
print( tensor.shape )
```

NumPy Indexing & Slicing

- NumPy indexing & slicing work in a similar way to lists, except that one can use n-dimensional indices

- Remember that Python indexing starts from 0

NumPy Indexing & Slicing

- Vectors

```
Vector = np.array( [10, 20, 30, 40, 50] )

# Getting a single element

print( Vector[0] ) # get the first element, answer will be 10

print( Vector[4] ) # get the fifth element, answer will be 50
```

NumPy Indexing & Slicing

- Vectors

 # Getting a series of elements

print(Vector[1:3]) # get all elements between second and fourth (fourth not included), answer will be [20, 30]

print(Vector[-2:]) # get the second last and last elements, , answer will be [40, 50]

print(Vector[0::2]) # get the even indices (0,2,4), answer will be [10, 30, 50]

print(Vector[1::2]) # get the odd indices (1,3), answer will be [20, 40]

NumPy Indexing & Slicing

- Vectors

 # Getting a series of elements

print(Vector[1:3]) # get all elements between second and fourth (fourth not included), answer will be [20, 30]

print(Vector[-2:]) # get the second last and last elements, , answer will be [40, 50]

print(Vector[0::2]) # get the even indices (0,2,4), answer will be [10, 30, 50]

print(Vector[1::2]) # get the odd indices (1,3), answer will be [20, 40]

NumPy Indexing & Slicing

- Matrices

```
Matrix = np.array( [ [10, 20], [30, 40] ] )

# Getting a single element

print( Matrix[0,0] ) # answer will be 10

print( Matrix[1,1] ) # answer will be 40
```

NumPy Indexing & Slicing

- Matrices

```
# Getting an entire row or column

print( Matrix[0,:] ) # first row, [10, 20]

print( Matrix[:,1] ) # second column [20, 40]
```

NumPy Indexing & Slicing

- Tensors

```
Tensor = np.array([[[10, 20, 30], [40, 50, 60]], [[100, 200, 300], [400, 500, 600]]])

# Getting a single element
print( Tensor[0,0,0] ) # answer will be 10
print( Tensor[0,0,1] ) # answer will be 20
print( Tensor[1,0,1] ) # answer will be 200
print( Tensor[1,1,2] ) # answer will be 600
```

NumPy Indexing & Slicing

- Tensors

```
#  Getting an entire row or column
print( Tensor[0,0,:] ) # first row of tensor at index=0, [10, 20, 30]
print( Tensor[0,:,0] ) # first column of tensor at index=0, [10, 40]
print( Tensor[1,:,0] ) # first column of tensor at index=1, [100, 400]
print( Tensor[1,:,2] ) # Third column of tensor at index=1, [300, 600]
```

Generating data with builtin functions

- One can use built-in functions within the NumPy library to generate standard matrices such as a matrices of ones, zeros, and identity matrix.

```
ones = np.ones( (3,3) )
print( ones )

zeros = np.zeros( (3,3) )
print( zeros )

identity_matrix = np.eye( 3 )
print( identity_matrix )
```

Generating data with built-in functions

- One can use the “random” function to generate pseudo-random numerical data

```
random_data_1 = np.random.rand(2,5)
random_data_2 = np.random.rand(2,5)
random_data_3 = np.random.rand(2,5,5)

# Fix random generation seed for reproducible data generation
np.random.seed(0)
random_data_4 = np.random.rand(2,2)
```

Generating data with built-in functions

- One can use the “arange” function to generate data within a certain range

```
vector_of_integers = np.arange( 0, 10 )
print( vector_of_integers )

vector_of_even_integers = np.arange( 2, 10, 2 )
print( vector_of_even_integers )

vector_of_odd_integers = np.arange( 1, 10, 2 )
print( vector_of_odd_integers )
```

Stacking data with NumPy

- Often, one may want to stack vectors or matrices (or tensors)

```
ones = np.ones( (3,3) )
zeros = np.zeros( (3,3) )

vert_stack = np.vstack( [ones, zeros] )
print( vert_stack.shape, vert_stack )

horiz_stack = np.hstack( [ones, zeros] )
print(horiz_stack.shape, horiz_stack )
```