CSCC11 – Tutorial 1

What is NumPy?

NumPy is a Python library that supports the creation and manipulation of large arrays and matrices, along with a wide range of functions that are useful for tasks such as data analysis and machine learning.

Installation:

- *Terminal:* Traverse into your working directory and run `pip install numpy`
- *Jupyter Notebook:* Run `%pip install numpy` or `!pip install numpy` in a cell

Import Convention: `import numpy as np`

One of the most commonly used data structures of this library is the NumPy array. It's similar to a Python list, but it stores elements of the same data type and it's more efficient for numerical computations.

Constructing an array:

Create a zero-filled array with `x` *rows and* `y` *columns*: `z = np.zeros((x, y))`

Create a one-filled vector of length `n`*:* `o = np.ones(n)`

Generate a 1-D array with all integers in [0, `n`)*:* `arr = np.arange(n)`

Generate a 1-D array with every `p` *values in* [`a, b`)*:* `arr = np.arange(a,b,p)`

Generate array and change dimensions to shape (x, y): `arr = np.arange(n).reshape(x, y)`

Create `n` x `n` *identity matrix:* `i = np.identity(n)` or `i = np.eye(n)`

Create `n` x `n` *diagonal matrix with* `v` *as its diagonal entries:* `d = v*np.eye(n)`

Create an empty array `n` *elements*: `e = np.empty(n)`

Create an array with `p` *equispaced points in* [`a, b`)*:* `arr = np.linspace(a,b,num=p)`

A few useful functions in NumPy:

Construct an array from a Python List: `arr = np.array(L)` where `L` is a Python list (can be a nested list [e.g., 2-D list] as well)

Append to an array: `arr = np.append(arr, to_append)` where `arr` and `to_append` are arrays

Concatenate two arrays `x` *and* `y` *along an axis* `a`*:* `c = np.concatenate((x, y), axis=a)`

Join a sequence of arrays `arrays` *along an axis* `a` *(default axis is 0):* `np.stack(arrays, axis=a)`

Delete i^{th} *element from an array:* `arr = np.delete(arr, i)`

Sort array in ascending order: `arr = np.sort(arr)`

Get datatype of entries in an array: `arr.dtype`

Get number of entries in an array: `arr.size`

Get number of array dimensions: `arr.ndim`

Get shape of array: `arr.shape` returns `(r, c)` where `r` is the number of rows and `c` is the number of columns

Expand the shape of an array along axis `a` *(default axis is 0):* `a = np.expand_dims(a, axis=a)`

Element-wise Addition of two arrays x *and* y*:* `x+y` or `np.add(x, y)`

Element-wise Subtraction of two arrays x *and* y*:* `x-y` or `np.subtract(x, y)`

Element-wise Product of two arrays x *and* y*:* `x*y` or `np.multiply(x, y)`

Element-wise Division of two arrays x *and* y*:* `x/y` or `np.divide(x, y)`

Multiplication of array x *by scalar* n*:* `n*x`

Transpose x*:* `x.T`

Dot product of two 1-D arrays x *and* y*:* `np.dot(x, y)`

Matrix multiplication of two matrices x *and* y*:* `np.matmul(x, y)`

Find the indices at which entries of array x *are non-zero:* `np.argwhere(x)`

Find the indices at which entries of array x *are greater than 1:* `np.argwhere(x>1)`

In-class Exercises:

1) Complete the code so that it generates the matrix below:

```
a = np.ones(_________)

b = np.concatenate(_________________, _________________)
```

$$b = \begin{bmatrix} 1 & 1 & 1 & 2 & 2 & 2 & 3 & 3 & 3 \\ 1 & 1 & 1 & 2 & 2 & 2 & 3 & 3 & 3 \\ 1 & 1 & 1 & 2 & 2 & 2 & 3 & 3 & 3 \end{bmatrix}$$

2) Without using loops, write Python code that generates an nxn matrix where the diagonal elements are zeros, and all other elements are ones. You can assume that n is initialized to some integer value greater than zero.

Exercise to be submitted:

Without using loops, write Python code that generates an nxn matrix containing all the integers in the range [1,n^2] sorted by row/column. You can assume that n is initialized to some integer value greater than zero. For example, assuming n is 3, the matrix would be as follows:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$