**CBE 5790 - Modeling and Simulation**

Worksheet #2 – Python basics with NumPy

**Deadline for uploading answers (pdf file) to Carmen:** Thu, Sep 6 at 2:20 PM

Use the NumPy package to work the exercises below. Import NumPy using the command:

import numpy as np

**Write a Python command to accomplish each task**

|  |  |
| --- | --- |
| calculate 1.2 sin(4π/7) e-3/4 | 1.2\*np.sin(4\*np.pi/7) \*np.exp(-3/4) |
| calculate 3 + 4ln(4) – 83log10(12) | 3 + 4\*np.log(4)-(8\*\*3)\*np.log10(12) |
| create matrices  and assign  to variables *x* and *y,* respectively | x=np.array([[1,3],[5,7]])  y=np.array([[0,2,4],[6,8,10]]) |
| transpose the matrix y | y.transpose() |
| calculate the square root of each element in *y* | np.sqrt(y) or np.power(y,1/2) |
| calculate the square of each element in *y* | np.square(y) or np.power(y,2) |
| perform the matrix multiplication of *y* and its transpose | np.matmul(y,y.transpose()) |
| perform the matrix multiplication *xy* (if possible) | np.matmul(x,y) |
| perform the matrix multiplication *yx* (if possible) | np.matmul(y,x)   * this gives an error |
| extract the 1st and 3rd columns of *y* and store as *z* | z=y[:,[0,2]] |
| multiply each element in *x* by the corresponding element in *z* | np.multiply(x,z) |
| divide each element in *x* by the corresponding element in *z* | np.divide(x,z) |
| concatenate the rows of matrices *x* and *z* | np.vstack([x, z]) |
| concatenate the columns of matrices *x*, *y*, and *z* | np.hstack([x, y,z]) |

**Describe the result of each Python command:**

|  |  |  |
| --- | --- | --- |
| Command | Description | Output |
| x = np.arange(-2,3,0.1) | Creates an array that starts with -2, stepsize 0.1 and ends at 2.9 |  |
| x = np.zeros((2,2)) | Creates a two by two array of zeros |  |
| 7\*np.ones((2,3)) | Creates a 2 by 3 array with ones and is multipled by 7, so it creates a 2 by 3 array of 7s | array([[7., 7., 7.],  [7., 7., 7.]]) |
| dog = np.array([[1, 2, 3],[4, 5, 6]]) | Created a 2 by 3 array (so the lists are stacked vertically) |  |
| np.dot(dog,dog) | Gives an error, because it’s a 2x3 dot 2x3 matrix | ValueError: shapes (2,3) and (2,3) not aligned: 3 (dim 1) != 2 (dim 0) |
| np.multiply(dog,dog) | multiplies each element in dog by the corresponding element in *dog* | Out[5]:  array([[ 1, 4, 9],  [16, 25, 36]]) |
| np.cos(np.multiply(np.pi,dog)) | Multiplies each term in dog matrix by pi and then the cosine of each term in the matrix was taken | array([[-1., 1., -1.],  [ 1., -1., 1.]]) |
| dog[1,0] | Displays the number in row 2, column 1 in matrix dog = 4 | Out[7]: 4 |
| dog[0] | Displays row 1 in matrix “dog” = [1 2 3] | Out[8]: array([1, 2, 3]) |
| dog.shape | Gives the entire array size: (2,3) | Out[9]: (2, 3) |
| dog.shape[0] | dog.shape = a tuple which is (2,3), so dog.shape [0] = 2 | Out[11]: 2 |
| dog.shape[1] | dog.shape = a tuple which is (2,3), so dog.shape [1] = 3 | Out[12]: 3 |
| np.mean(dog) | Gives average of matrix | 3.5 |
| np.mean(dog,axis = 0)  \*axis=0 refers to column | Gives mean of each COLUMN in matrix of dog and Creates an array | array([2.5, 3.5, 4.5]) |
| np.mean(np.transpose(dog), axis = 0) | Transposes dog matrix to 3x2 matrix and then takes the average of each COLUMN and creates an array | array([2., 5.]) |
| np.sum(dog, axis = 1)  \*axis=1 refers to row | Creates an array that has the sum of each ROW | array([ 6, 15]) |
| dog[0,:] | Creates array of row 1 and all columns | array([1, 2, 3]) |
| dog[:,0] | Creates array of all rows and first column | array([1, 4]) |
| dog[1,1:3] | Creates an array that consists of row 2 and column 2 to 3 from dog matrix | array([5, 6]) |
| dog[1,[0,2]] | Creates an array that consists of row 2 and column 1, 3 from dog matrix | array([4, 6]) |
| dog[1,-1] | This gives a number from dog’s season row 2 column 3 | 6 |
| np.concatenate((dog, np.array([[0],[9]])),axis = 1) | Concatenates the dog matrix and a new column | array([[1, 2, 3, 0],  [4, 5, 6, 9]]) |
| print(np.abs(-9) == np.power(-3,2)) | Sets absolute value of (-9) equal to (-3)^2 and asks it to yield either a True or False Statement: yields True | True |
| print(np.abs(-9) == -3\*\*2) | Sets absolute value of (-9) equal to -3^2 and asks it to yield either a True or False Statement: yields false, bc  -3^2=-9 | False |
| print(np.abs(-9) == (-3)\*\*2) | Sets absolute value of (-9) equal to (-3)^2 and asks it to yield either a True or False Statement: yields True, bc  (-3)^2=9 | True |
| print(np.sin(True)) | True is the same thing as “1” and sin(1) = 0.84147 | 0.8413 |
| x = 21  print(np.cos(x\*False)) | 1 | 1 |
| print(np.logical\_xor(x>0, x<30)) | “xor” means that ONLY ONE of those statements can be true for entire thing to be true. If both statemnets are true, then the entire thing is false | False |

**Given x = np.array([2, 6, -3, 5, -1]) and y = np.array([[1, 2, 3], [4, 5, 6]]) describe the result of the code:**

|  |  |  |
| --- | --- | --- |
| Command | Description | Output |
| print(np.where(x==-3)) | Will give an array of location in array x where -3 is which is position 2 | (array([2], dtype=int64),) |
| print(np.where(x>0)) | Will give an array of location in array x where –the value is greater than 0: [0,1,3] | (array([0, 1, 3], dtype=int64),) |
| print(np.where(x==4)) | Will give an array of location in array x where –the value is equal to 4: empty array | (array([], dtype=int64),) |
| print(x==4) | Compares 4 to every term in the x array, and gives an array of true or false words. Since there is no 4 in x array, the array will give 5 false words | [False False False False False] |
| print(x>3 and x<0)  \*not bit wise | Compares every the ENTIRE x array as A WHOLE to 3 and 0 and gives an error | ValueError: The truth value of an array with more than one element is ambiguous. Use a.any() or a.all() |
| print((x>3) & (x<0))  \*bit wise | Compares EVERY TERM term in x array to 3 and 0 and gives and array to True and False words: | [False False False False False] |
| print(x[x<0]) | Will give an array of the actual values in matrix x that are less than 0 | [-3 -1] |
| k = x>2  z = np.logical\_not(k) | Compares 2 to every term in the x array, and gives an array of true or false words. The logical\_not gives the opposite word |  |
| print(np.mean(x[x>4])) | Will give an array of the actual values in matrix x that are more than 4 and takes the average of those values | 5.5 |
| k = x<0  np.sum(x[k]) | These two commands together will yield a array of the actual values in matrix x that is less than zero and then adds the values together. | -4 |
| k = [0, 0, 1, 0, 1]  z = x[k] | Gives the value of k position in matrix x |  |
| k = np.array([True, True, False, False, True])  z = x[k] | Gives an array of the values in x array that corresponds to the True position in the k array |  |
| print(np.any(y>4)) | See if there is ATLEAST one value in the y matrix is greater than 4 and then yields a single True or False statement | True |
| print(np.any(y>4, axis = 0)) | See if there is atleast one value in EACH and ALL COLUMNS in y matrix is greater than 4 and then yields a 3x1 array of True and False statement | [False True True] |
| print(np.any(y>4, axis = 1)) | See if there is atleast one value in EACH and ALL ROWS in y matrix is greater than 4 and then yields a 2x1 array of True and False statement | [False True] |
| print(np.all(x>-5)) | See if ALL values in the x matrix is greater than 5 and then yields a single True or False statement | True |
| print(np.all(x>0)) | See if ALL values in the x matrix is greater than 0 and then yields a single True or False statement | False |