Numpy Exercises

This notebook contains the exercises for question 3. Please fill out the incomplete cells, and attach this notebook as a pdf in your write-up when you are done

In [2]:
```
import numpy as np
```

We've provided sample test cases for you to verify your code with. Please note that passing these test cases does **not** guarantee that your implementation is correct. We encourage you to write your own test cases, especially when debugging.

In [3]:
```
TEST_INPUT_PART_A = np.array([[0.37454012, 0.95071431, 0.73199394],
 [0.59865848, 0.15601864, 0.15599452],
 [0.05808361, 0.86617615, 0.60111501],
 [0.70807258, 0.02058449, 0.96990985],
 [0.83244264, 0.21233911, 0.18182497]])
TEST_OUTPUT_PART_A = np.array([2.05724837, 0.91067164, 1.52537477, 1.6985669

TEST_INPUT_PART_B1 = np.array([1, 2, 3, 4])
TEST_INPUT_PART_B2 = np.array([0.1, 0.2, 0.3])
TEST_OUTPUT_PART_B = np.array([[1.1, 1.2, 1.3],
 [2.1, 2.2, 2.3],
 [3.1, 3.2, 3.3],
 [4.1, 4.2, 4.3]])

TEST_INPUT_PART_C1 = np.array(
[[0.00552212, 0.81546143, 0.70685734],
 [0.72900717, 0.77127035, 0.07404465],
 [0.35846573, 0.11586906, 0.86310343],
 [0.62329813, 0.33089802, 0.06355835]]
)
TEST_INPUT_PART_C2 = np.array(
[[0.31098232, 0.32518332, 0.72960618],
 [0.63755747, 0.88721274, 0.47221493],
 [0.11959425, 0.71324479, 0.76078505],
 [0.5612772,  0.77096718, 0.4937956 ],
 [0.52273283, 0.42754102, 0.02541913],
 [0.10789143, 0.03142919, 0.63641041]]
)
TEST_OUTPUT_PART_C = np.array([2, 1, 0, 1, 3, 2])

TEST_INPUT_PART_D0a = np.array([[0.63352971, 0.53577468, 0.09028977, 0.83530
 [0.32078006, 0.18651851, 0.04077514, 0.59089294],
 [0.67756436, 0.01658783, 0.51209306, 0.22649578],
 [0.64517279, 0.17436643, 0.69093774, 0.38673535],
 [0.93672999, 0.13752094, 0.34106635, 0.11347352]])
TEST_INPUT_PART_D0b = np.array([[0.92469362],
 [0.87733935],
 [0.25794163],
 [0.65998405]])
```

```
TEST_OUTPUT_PART_D0 = np.array([2, 0.719627281044947])

def check_answer(predicted, actual):
    try:
        assert np.allclose(predicted, actual), "INCORRECT"
        print("CORRECT")
    except:
        print("INCORRECT")
```

Part A (1 pt)

One use case of numpy is computing the norm of a set of vectors. In this question, given $N×M$ matrix A compute numpy array B of size N such that entry $B[i]$ is the l_1-norm of row i in matrix A (maximum lines of code 1 - note that line limits do not include the function definition lines; no partial credit will awarded if more lines are used).

```
In [4]: def part_a(A):
            return np.fromfunction(lambda i, j: np.linalg.norm(A,ord=1,axis=1), (5,1
            pass
        check_answer(part_a(TEST_INPUT_PART_A), TEST_OUTPUT_PART_A)
```

```
CORRECT
```

Part B (2 pts)

Another useful feature in numpy is broadcasting. Sometimes given a pair of vectors (or more) we wish to reconstruct a matrix from them. In this case given a numpy array A of size N and a numpy array B of size M, construct and return $N \times M$ matrix C where $C[i, j] = A[i] + B[j]$. (maximum lines of code 1 - note that line limits do not include the function definition lines; no partial credit will awarded if more lines are used).

```
In [5]: def part_b(A, B):
            return np.fromfunction(lambda i, j: A[i]+B[j], (len(A), len(B)), dtype =
            pass

        check_answer(part_b(TEST_INPUT_PART_B1, TEST_INPUT_PART_B2), TEST_OUTPUT_PAR
```

```
CORRECT
```

Part C (3 pts)

Another potential application is assigning points to groups. In this question we will consider a set of cell towers and a set of home addresses. The goal is to find the the closest cell tower for each home address. You are given the set of cell towers in matrix A. A is a $M \times D$ matrix which denotes the locations of M towers in a D dimensional space. You are also given a $N \times D$ matrix B which is the set of locations of the home addresses. You must return numpy array C of size N where

$C[i]$ is the cell tower that home i (i^{th} row of matrix B) should be assigned to. For example, if the 9th home address $B[9, :]$ is closest to the 6th cell tower $A[6, :]$ then $C[9] = 6$. (maximum lines of code 4 - note that line limits do not include the function definition lines; no partial credit will awarded if more lines are used).

HINT: $$||\bar{x} - \bar{y}||^2 = ||\bar{x}||^2 + ||\bar{y}||^2 - 2*\bar{x}^T\bar{y}$$

For this question, we have provided a buggy solution. Please find and fix the bugs in the solution for full credit.

```
In [6]: def part_c(A, B):
            C = np.zeros((len(B), 1))
            for a in range(0, len(B)):
               C[a] = np.argmin(np.fromfunction(lambda i, j: np.linalg.norm(A, axis=
            return C.T

        check_answer(part_c(TEST_INPUT_PART_C1, TEST_INPUT_PART_C2), TEST_OUTPUT_PAR
```

```
CORRECT
```

Part D (3 pts)

Given M D-dimensional vectors in matrix A and vector B, find the vector in A with the smallest $\cos{\alpha}$ (where α is the angle between the vector in A and B) and return as a tuple (the index of this vector, $\cos{\alpha}$ between them). In this case, matrix A is an $M \times D$ matrix of points in D dimensional space and vector B is a $D \times 1$ vector. For example if it is the vector in row 6, return (6, cos(angle)).You may assume the angle for all vectors is between ($\frac{-\pi}{2}$ and $\frac{\pi}{2}$ degrees). (maximum lines of code 5 - note that line limits do not include the function definition lines; no partial credit will awarded if more lines are used).

HINT: $$\bar{a} \cdot \bar{b} = ||\bar{a}|| ||\bar{b}|| cos(\alpha)$$

```
In [7]: def part_d(A, B):
            C = np.zeros((len(A),1))
            for a in range(0,len(A)):
                C[a] = np.matmul(A[a],B) / (np.sqrt(np.matmul(A[a],A[a].T)) * np.sqr
            return np.array([np.argmin(C), np.min(C)])

        check_answer(part_d(TEST_INPUT_PART_D0a, TEST_INPUT_PART_D0b), TEST_OUTPUT_F
```

```
CORRECT
```

Make sure to export this notebook as a pdf and attach it to your solution document.