Assignment 2 (Markov Chains, 35 marks) You can assume that all inputs are given in the correct format. I.e. you don't need to include checks for the correct input format (but need to make sure during the code testing that this is the case).

YOU CANNOT USE PRE-DEFINED FUNCTIONS IN PYTHON FOR EXERCISE 1, I.E. YOU CANNOT USE FUNCTIONS THAT CARRY OUT MATRIX MULTIPLICATION OR MATRIX POWERS FROM PYTHON. INSTEAD, WRITE THESE FUNCTIONS *FROM SCRATCH* USING THE USUAL DATA STRUCTURE OF MATRICES AS LISTS OF LISTS. *

SUBMIT THIS ASSIGNMENT THROUGH CANVAS.

Q 1 (10 marks)

Write a Python function def matrixMultiplication(M1,M2) that computes the result of multiplying two matrices M1 and M2. To ensure that the multiplication makes sense, you can assume that the number of columns (i.e the length of the first row of input M1) is identical to the number of rows of input M2. (5 marks)

Initialize a variable named **res** with a matrix consisting of zero entries. This matrix should be a k x l matrix in case M1 is a k x m matrix and M2 an m x l matrix. Use nested for-loops to compute the matrix multiplication, following the standard definition of matrix multiplication. The function needs to return the value of a variable res (containing the final result of the matrix multiplication).

Write a function powerOfMatrix(M,n) to compute the n-th power of a square matrix M (n is a positive integer, greater than or equal to 1). Store the result of the computation in a variable P which needs to be returned. You can rely on the solution for part a) to complete the exercise.
(5 marks)

Q 2 Markov Chains (15 marks)

A) **10 marks** Write a Python function markov_chain_path(transitionMatrix, time, initial_state) transitionMatrix determines a Markov chain starting from the initial_state at time 0 the function markov_chain_path must compute a Markov chain path, starting from the given initial state.

Given a square transitionMatrix, say M = [[0, 1, 0], [0.5, 0, 0.5], [1, 0, 0]]

The Markov chain corresponding to this transition matrix has 3 states, namely: 1, 2 and 3 (where the states are represented by positive integers) The matrix element in position i, j (i.e. located in row i, column j of the transition matrix) thus represents the transition from state i +1 to state j + 1 (where we use the integer notation for states introduced above). Store the possible state transitions in a variable named: transitions. These state transitions are represented as a list. For instance, for the given matrix M this list is: [[[1, 1], [1, 2], [1, 3]], [[2, 1], [2, 2], [2, 3]], [[3, 1], [3, 2], [3, 3]]]

Start your file by importing the following:

```
import numpy as np
import random as rm
```

given a transition matrix, use the following function to generate random transitions from a state i to a state j (which is reachable in 1 step from i via an appropriate transition from i to j). The random transition is stored in a variable named: choice

```
def random_transition_choice(i, transitions, transitionMatrix_size):
        choice = np.random.choice(range(transitionMatrix_size),
```

replace = True,
p = transitionMatrix[i]) return transitions[i][choice] where: transitionMatrix_size = len(transitionMatrix[0])

Write a function

markov_chain_path(transitionMatrix, time, initial_state)

which takes as inputs a transition matrix, time (a positive integer N greater than or equal to 0) and an initial state, represented as a 1 x transitionMatrix_size matrix.

The initial state represents a single state from which we start the Markov chain via a binary list. The binary list contains only one non-zero entry (i.e. the entry 1 at the position of the corresponding initial state).

The function needs to use

random_transition_choice(i, transitions, transitionMatrix_size)

to compute a path of this Markov chain for the duration of a given time. Store the states encountered along the path in a variable named: states_path Compute the probability of this path.

The program needs to print the following three statements:

- the value of states_path, preceded by the string:
  "States taken at each time:"
- the final state on the path (for the given time and computed path), preceded by the string:
  "Final state at time " + str(time) + ": "
- the probability of the state path, preceded by the string:
  "Probability of the state path taken: "

B) **Markov chain convergence exercise (5 marks)**

Write a Python function markov_chain_convergence_test(transitionMatrix, time, initialDistribution) initialDistribution stores the probabilities for each state at time 0 in a 1 x K matrix, where K is the number of states. If there are three states, with probabilities 0.5, 0.2 and 0.3, then initialDistribution is the matrix: [[0.5, 0.2, 0.3]] (containing one row of length 3) The function needs to compute the n-th power of the transitionMatrix where n is the positive integer value stored in the variable time1. The result is stored in a variable named: power The function needs to store the result of multiplying the matrices initialDistribution with power in a variable named: result and print result. Test the function for input matrix M1 = [[0, 1, 0], [0.5, 0, 0.5], [1, 0, 0]] Use the initial distribution [[1, 0, 0]] (starting at state 1) Compute and print the outputs for times 1, 10, 50 and 100.

Q 3 Random Walk (10 marks)

Write a Python function random_walk(n) that takes as input a positive integer n indicating the number of steps the walk takes. The walk occurs in the 2 dimensional grid with integer coordinates. The walk can go North ('N'), South ('S'), East ('E') or West ('W'), indicated by the characters displayed. Use the imported random library to implement the walk.