Lab 5: Markov Chain Lab

This is an INDIVIDUAL assignment. Due date is as indicated on BeachBoard. Follow ALL instructions otherwise you will lose points. In this lab, you will be finding the probability of a future outcome based on some current state. Unlike the previous lab, you should use the numpy library for this lab.

Background:

A Markov chain is a process where future outcomes can be predicted by a current state. These future outcomes are based on probabilities. The probability of moving on to a certain state depends on the previous state; this probability does not change with time. (Please note that this is a VERY oversimplified explanation. There's so much more to this concept.)

We can express these probabilities as a transition matrix that looks like below:

$$P = \begin{bmatrix} p_{00} & p_{01} & \cdots & p_{0n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{n0} & p_{n1} & \cdots & p_{nn} \end{bmatrix}$$

Where p_{ij} is the probability of moving from state i to state j.

An example of a Markov chain is the weather of one day based on the previous day where the states are (1) sunny, (2) cloudy, (3) rainy.

Weather	Weather of next day		
	Sunny (j=0)	Cloudy (j=1)	Rainy (j=2)
Sunny (i=0)	80%	18%	2%
Cloudy (i=1)	40%	50%	10%
Rainy (i=2)	20%	60%	20%

This table represents the probabilities of some day's weather based on the previous day's weather. For example, the probability of it being sunny the day after a cloudy day (p_{10}) is 40%.

We can get P^n (nth power of P) by matrix multiplication. P^n would represent the probability of passing from state i to state j in n stages.

$P^1 = \begin{bmatrix} p_{00} & p_{01} & \cdots & p_{0n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{n0} & p_{n1} & \cdots & p_{nn} \end{bmatrix}$ would represent the probabilities after 1 stage

$P^2 = \begin{bmatrix} p_{00} & p_{01} & \cdots & p_{0n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{n0} & p_{n1} & \cdots & p_{nn} \end{bmatrix} \times \begin{bmatrix} p_{00} & p_{01} & \cdots & p_{0n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{n0} & p_{n1} & \cdots & p_{nn} \end{bmatrix}$ would represent the probabilities after 2 stages

$P^3 = \begin{bmatrix} p_{00} & p_{01} & \cdots & p_{0n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{n0} & p_{n1} & \cdots & p_{nn} \end{bmatrix} \times \begin{bmatrix} p_{00} & p_{01} & \cdots & p_{0n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{n0} & p_{n1} & \cdots & p_{nn} \end{bmatrix} \times \begin{bmatrix} p_{00} & p_{01} & \cdots & p_{0n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{n0} & p_{n1} & \cdots & p_{nn} \end{bmatrix}$ 3 stages

In our weather example, (your input matrix will be a percentage, you must change it to decimals using `per_to_dec()`)

$P^1 = \begin{bmatrix} 0.8 & 0.18 & 0.02 \\ 0.4 & 0.5 & 0.1 \\ 0.2 & 0.6 & 0.2 \end{bmatrix}$ after 1 stage (next day's weather)

$P^2 = \begin{bmatrix} 0.8 & 0.18 & 0.02 \\ 0.4 & 0.5 & 0.1 \\ 0.2 & 0.6 & 0.2 \end{bmatrix} \times \begin{bmatrix} 0.8 & 0.18 & 0.02 \\ 0.4 & 0.5 & 0.1 \\ 0.2 & 0.6 & 0.2 \end{bmatrix} = \begin{bmatrix} 0.716 & 0.246 & 0.038 \\ 0.54 & 0.382 & 0.078 \\ 0.44 & 0.456 & 0.104 \end{bmatrix}$ after 2 stage (after 2 days)

Thus, the probability of it being sunny two days after a cloudy day (P^2_{10}) is 54%

If you continue to calculate P^1, P^2, P^3, P^4, P^5 … , you will find the matrix stops changing (changes negligibly). This is called the long-run distribution.

```
P2=
[[0.716 0.246 0.038]
 [0.54  0.382 0.078]
 [0.44  0.456 0.104]]
P3=
[[0.6788  0.27468 0.04652]
 [0.6004  0.335   0.0646 ]
 [0.5552  0.3696  0.0752 ]]
P4=
[[0.662216 0.287436 0.050348]
 [0.62724  0.314332 0.058428]
 [0.60704  0.329856 0.063104]]
P5=
[[0.6548168  0.29312568 0.05205752]
 [0.6392104  0.305126   0.0556636 ]
 [0.6301952  0.3120576  0.0577472 ]]
P6=
[[0.65151522 0.29566438 0.05282041]
 [0.64455144 0.30101903 0.05442953]
 [0.64052864 0.30411226 0.0553591 ]]
P7=
[[0.650042   0.29679717 0.05316082]
 [0.64693467 0.29918649 0.05387884]
 [0.64513964 0.30056675 0.05429362]]
P8=
[[0.64938464 0.29730264 0.05331272]
 [0.6479981  0.29836879 0.05363311]
 [0.64719713 0.29898468 0.05381819]]
P9=
[[0.64909131 0.29752819 0.0533805 ]
 [0.64847262 0.29800392 0.05352346]
 [0.64811521 0.29827874 0.05360605]]
P10=
[[0.64896042 0.29762883 0.05341075]
 [0.64868435 0.29784111 0.05347454]
 [0.64852488 0.29796374 0.05351139]]
P11=
[[0.64890202 0.29767374 0.05342424]
 [0.64877883 0.29776846 0.05345271]
 [0.64870767 0.29782318 0.05346915]]
P12=
[[0.64887596 0.29769378 0.05343026]
 [0.64882099 0.29773604 0.05344296]
 [0.64878924 0.29776046 0.0534503 ]]
```

Notice that all values from P^{11} to P^{12} have a difference of less than 0.0001. This is where we will stop for this lab to find our "long-run-distribution"

Instructions:

1. Take a close look at the `Markov.py` file. There are four empty functions:

`per_to_dec(mat)`: Converts percentages in matrix into a decimals
`sig_change(oldmat, newmat)`: Checks if there are any "significant" changes between spatially corresponding elements in a matrix. (use this function in `long_run_dist`) **You are not allowed to use the isclose() or any() or all() function. You will not get any points if you do.**
`prob_x(mat, x)`: Returns the transition probability matrix after x stages. **You are not allowed to use the matrix_power() function. You will not get any points if you do.**
`long_run_dist(probs)`: Returns the long-run-distribution. **You are not allowed to use the matrix_power() or isclose() or any() or all() function. You will not get any points if you do.**

Read through both of their descriptions carefully. Remember, you will lose points if you do not follow the instructions. We are using a grading script

2. Your job is to implement all of the listed functions so that it passes any test case. There are sample test cases provided for you, but these are not the only cases that we will test. Remember, we will be testing each function separately and collectively. They should be able to work by themselves and harmoniously with other functions.

3. After completing these functions, comment out the test cases (or delete them) or else the grading script will pick it up and mark your program as incorrect. Also, MAKE SURE THAT YOUR PROGRAM DOES NOT PRINT ANYTHING!!! The grading script may pick up any printed text and misinterpret it as a test case or input causing you to lose points.

4. Convert your `Markov.py` file to a `.txt` file. Submit your `Markov.py` file and your `.txt` file on BeachBoard. Do NOT submit it in compressed folder.

5. Do not email us your code asking us to verify it. We will answer general questions, but we will not debug your code over email.

Grading rubric:

To achieve any points, your submission must have the following. Anything missing from this list will result in an automatic zero. NO EXCEPTIONS!

- Submit both py and txt files
- Files named correctly
- Program has no errors (infinite loops, syntax errors, logical errors, etc.) that terminates the program

Please note that if you change the function headers or if you do not return the proper outputs according to the function requirements, you risk losing all points for those test cases.

Points	Requirement
5	Correct submission: py and txt file
5	Implemented `per_to_dec()` correctly. (5 test cases)
15	Implemented `sig_change()` correctly. (5 test cases)
10	Implemented `prob_x()` correctly. (5 test cases)
15	Implemented `long_run_dist()` correctly. (5 test cases)
5	Passes 5 original test cases (also commented out or deleted the test cases)

***** NOTHING should be rounded. You may lose points if your values are rounded**