Instructions

- In each problem you are asked to write a computer program to calculate mathematical quantities related to the course content. In addition to your final program, you should have a clear mathematical explanation backing up how your program works. A correct program that is not well explained will not receive full marks. You should also make your program as efficient as possible using numpy vector functions (see grading scheme for details!)
- All answers should be exact up to computer precision unless otherwise specified. (That is, if it says "calculate x" you should calculate the exact value of x up to the number of decimals the computer uses, not a Monte Carlo estimate for x.)

0.1 Problem 1 - Simulating a Markov chain using uniform $[0,1]$ random variables

Consider a Markov chain on the state space $S = \{0, 1, \ldots, N_{state} - 1\}$ with an $N_{state} \times N_{state}$ transition matrix P. Write a program that inputs the transition matrix P (as a numpy array of shape (N_state,N_state), an initial state $x_0 \in S$, a random uniform variable $U \in (0,1)$ and returns a random state $X_1 \in S$ so that X_1 is a sample of the Markov chain at time 1. For full efficiency marks, minimize the number of if statements/for loops that are needed by using numpy vector functions instead. (Hint: Its possible to use make a version of this that uses the numpy function "searchsorted")

0.2 Problem 2 - A weather model

On any given day, it is either rainy or sunny. A model for the weather is the following rules which are applied in order (i.e. rule 1 is applied if possible; if it is does not apply, then rule 2 is applied etc.)

1. If it is sunny today and it was sunny yesterday, then the forecast for the next day is50% rainy-50% sunny.
2. If it is sunny today, but it was rainy yesterday, then the forecast for the next day is 30% rainy - 70% sunny
3. Otherwise, if has been rainy for three days in a row, the forecast for the next day is 10% rainy - 90% sunny.
4. Otherwise, if has been rainy for two days in a row, the forecast for the next day is 25% rainy - 75% sunny.
5. Otherwise, if it is rainy today, the forecast for the next day is 40% rainy - 60% sunny.

a) Find a Markov chain to model this problem. Be sure to clearly indicate the state space you are using is, what the transition matrix is and draw the corresponding directed graph of the Markov chain.

b) Write a program that inputs an integer n and outputs the probability that it is rainy n days from now given that it is sunny today. Draw a graph of this probability as function of n for $n \in [0, 100]$.

c) Calculate the exact value of

$$\lim_{n\to\infty} \mathbf{P}\left(\text{Rainy on day } n \,|\text{Sunny today and sunny yesterday}\right)$$

0.3 Problem 3 - Monkey at a typewriter (also known as the "ABRACADABRA" problem)

A monkey is typing on a typewriter and generates a long string of random letters. For example, the first 14 characters might look like the monkey might type the string:

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	14
Letter	B	C	D	B	A	B	R	A	C	A	D	A	B	R	A

We are interested in investigating how long it takes for interesting words to appear in the monkey's text. For example, in the above example, the words "ABRACADABRA" fully appears in the text after 14 letters have been typed.

We model this situation as follows:

- Assume there are $n_{buttons}$ possible buttons on the typewritter the monkey can press which are ordered $b_0, b_1, \ldots b_{n-1}$. Any word can be represented by a finite sequence of integers. For example, if $b_0 =$ "A", $b_1 =$ "B", $b_2 =$ "C",$b_3 =$ "D", $\ldots b_{18} =$ "R" then the word "ABRACADABRA" is the sequence $0, 1, 18, 1, 2, 0, 3, 0, 1, 18, 0$. On a computer we represent this as a vector of length n_{word}. Each element of the vector is an integer in $[0, n_{word} - 1]$)
- Assume all buttons are equally likely.
- Assume the button pressed is independently at random, independent of all the other button presses that have ever been pressed.

You will write a program that inputs the **number of buttons** $n_{buttons}$ and a **word sequence** $\vec{w}$ (given as a vector of length n_{word} consisting of integers in $[0, n_{buttons} - 1]$) and uses Markov chains to output the **expected amount of time** until the word sequence $\vec{w}$ appears. Before you write the program, do the following:

- a) A naive way to model this as a Markov chain would be to use the sequence of the last n_{word} letters that appear in the monkey's text (e.g. the state space is the set of ALL possible n_{word} letter words). Explain why using this as the Markov chain would be difficult for the computer to handle.
- b) Instead of the naive way above, come up with a Markov chain for the problem which the size of the state space does not depend on $n_{buttons}$ that can be used to solve this problem. Be sure to clearly indicate the states of the Markov chain!
- c) Analyze the simple case of $n_{buttons} = 6$ by hand (This corresponds to the situation where the monkey's typing is the same as rolls of fair 6 sided dice)
  - Draw out the directed graph of the Markov chain corresponding to the word sequence "$0, 0$"
  - Draw out the directed graph of the Markov chain corresponding the word sequence "$0, 1$"
  - Using the graphs, do you think the expected amount of time for the monkey to type these two word sequences is same? Give an intuitive explanation why or why not.
- d) Write the general program!

PS Its possible to find a nice analytical formula for this expectation, but the proof involves something called the optional stopping theorem which is outside of generic Markov chain theory. We want the solution that uses Markov chains!

0.4 Problem 4 - Generalized PIG

(See the posted videos and other content about the dice game PIG or this question won't make sense) In the generalized version of the dice game PIG we have the following rules:

- On each roll, there is a probability p_{bust} of going bust and losing all our runner progress.
- With probability $1 - p_{bust}$ we do not go bust. When this happens we advance our run by some random non-negative integer. The distribution of this advancement is given by a probability vector $\vec{v} = (p_0, p_1, \ldots, p_{n_{max}}) \in \mathbb{R}^{n_{max}+1}$ where n_{max} is the maximum possible we can advance, and v_i is the probability we advance our runner i places. (Note that $v_0 > 0$ is possible)

(In classic pig with a 6 sided dice, $p_{bust} = \frac{1}{6}$, and $\vec{v} = \left(0, 0, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}\right) \in \mathbb{R}^{6+1}$)

Let s be an integer, and consider the "stop at s strategy" where we reroll our dice if our runner has advanced less that s spaces.

a) There is a simple way to find the value s^* that maximizes the expected score of this strategy over all possible s. Using math (no program) find a simple formula for s^* in terms of p_{bust} and $\vec{v}$. (Hint: Watch the numberphile video on PIG!). Also explain why when playing PIG always using the strategy to stop at s^* is not optimal! (Hint: Watch the numberphile video on PIG!)

b) Using Markov chains, write a program that inputs s, p_{bust} and $\vec{v}$ and outputs the expected score playing the stop at s strategy in generalized PIG and the probability to go bust at any time playing this strategy.

0.5 Problem 5 - "Can't Stop" simple roll_again strategy

(This problem does not involve Markov chains, but instead is about the final project game Can't Stop and is part of the work needed to make a good Can't Stop AI. Make sure you watch the videos/rules for Can't Stop or this question won't make sense)

a) Write a program that inputs the position of the runners (the runner positions are encoded as a vector of size $(11,)$) and the set of illegal columns (encoded as a boolean vector of size $(11,)$) and outputs the probability p_{bust} of going bust if you choose to reroll the dice. (HINT: You are given a function "prob_to_miss_all_targets" that calculates a certain probability of dice pairs missing a given list of columns....using this function will make part a) a lot easier!)

b) Relate the decision of "Reroll again or don't" in Can't Stop and the probabilities you calculated in a) to the dice game PIG. Use this to come up with a simple AI algorithm that decides whether or not roll_again in "Can't Stop". Your AI will be compared to the "random AI" that choose whether or not to roll again randomly. (You should be able to easily beat this AI!)

PS This question is more open ended than the all the other questions; you will have to make some simplifying assumptions to get to something that has a clear solution. Be sure to clearly explain your thinking and any simplifying assumptions you made. There is no single "right" answer here!