ASSIGNMENT COVERSHEET

PRAGUE CITY UNIVERSITY

Student Name: Hector David Peralta Ramirez	
Class: Programming Fundamentals	
Assignment: Markov chain and simulation with Python - ICA2	
Lecturer: Gramoz Cubreli	Semester: 1
Due Date: January 19, 2024	Actual Submission Date: Submission date

Evidence Produced (List separate items)	**Location (Choose one)**	
	X	Uploaded to the Learning Center (Moodle)
		Submitted to reception
Note: Email submissions to the lecturer are not valid.		

Student Declaration:	
I declare that the work contained in this assignment was researched and prepared by me, except where acknowledgement of sources is made. I understand that the college can and will test any work submitted by me for plagiarism. **Note:** The attachment of this statement on any electronically submitted assignments will be deemed to have the same authority as a signed statement	
Date: January 19, 2024	Student Signature: Hector David Peralta Ramirez

A separate feedback sheet will be returned to you after your work has been graded.
Refer to your Student Manual for the Appeals Procedure if you have concerns about the grading decision.

Student Comment (Optional)
Was the task clear? If not, how could it be improved?
Was there sufficient time to complete the task? If not, how much time should be allowed?
Did you need additional assistance with the assignment?
Was the lecturer able to help you?
Were there sufficient resources available?
How could the assignment be improved?

Markov chain and simulation with Python - ICA2

Hector David Peralta Ramirez

January 19, 2024

Contents

1 Introduction

A Markov chain was a discovery very important in the field of probability, its considered a "mathematical system that experiences transitions from one state to another according to certain probabilistic rules. The defining characteristic of a Markov chain is that no matter how the process arrived at its present state, the possible future states are fixed"(Henry Maltby 2023). Markov chain is also defined as an stochastic process but in contrast to a generic stochastic process a Markov chain needs to be without memory, that is, the steps that lead to the current condition do not determine future actions this is the Markov property.

2 Applications

Markov chain has a lot of applications that is why is so versatile and revolutionary, **healthcare** is one of the fields were Markov chains had been applied analyzing infertility treatments, the Markov chains can give a probability of a successful pregnancy following a series of infertility therapies, helping to understand how each stage of the therapy affects the probability of a successful pregnancy. Also another field were Markov chain can be applied is on **weather modeling**, Markov chain are applied to construct models for weather forecasting using transition matrices, this matrices are used for prediction giving an effective solution, also it is used in more complex problems, By creating a transition probability matrix, evaluating, and computing with a Markov chain, a mathematical model of weather and market forecasts can be built. This two applications are more focused in the science field but the Markov chain can also be applied in other terms like in the **stock market**, this sector is very versatile and constantly fluctuating and has a lot of impact in the economy, a lot of people like to invest in it that is why there exist a lot of methods that try to predict the future stock prices, and one of this methods is the Markov chain using "four steady states—variables that represented the possibility that a stock price on a given day would fall into one of the four states were obtained by solving a system of equations using a 4x4 transitional probability matrix"(Mohite, Wadkar, and Zanjurne 2023). By using this information with the real data, it is possible to forecast the next stock prices for the upcoming months.

This are only a few applications that the Markov chain can be used for, the purpose of explaining these applications is to understand the importance of the probability and the Markov chain effects in other fields and that nowadays is still a very important for science, finance, weather modeling, engineering and so on.

3 Markov Chain Modeling in Regeneration of the Axolotl

Markov chains have been used for multiple things over the years, for this project it was decided to apply it for the biological approach using two main subjects for this project, which are humans and axolotls, the use of Markov chains . to simulate regenerative processes is interesting for mathematical and programming models. Ambystoma mexicanum, the scientific name of axolotls, is very interesting for this research since they have an impressive regenerative capacity in both tissues and limbs. The project will assign states that will represent the regenerative phase in which each one is (Human and axolotl) and the transition matrices will be those that will indicate the possibility of advancing in this regeneration process. "scientists study the genetic and biochemical mechanisms that drive axolotl tissue regeneration in hopes that deeper understanding may bridge the gap between regenerative biology and medicine"(NIH 2021).

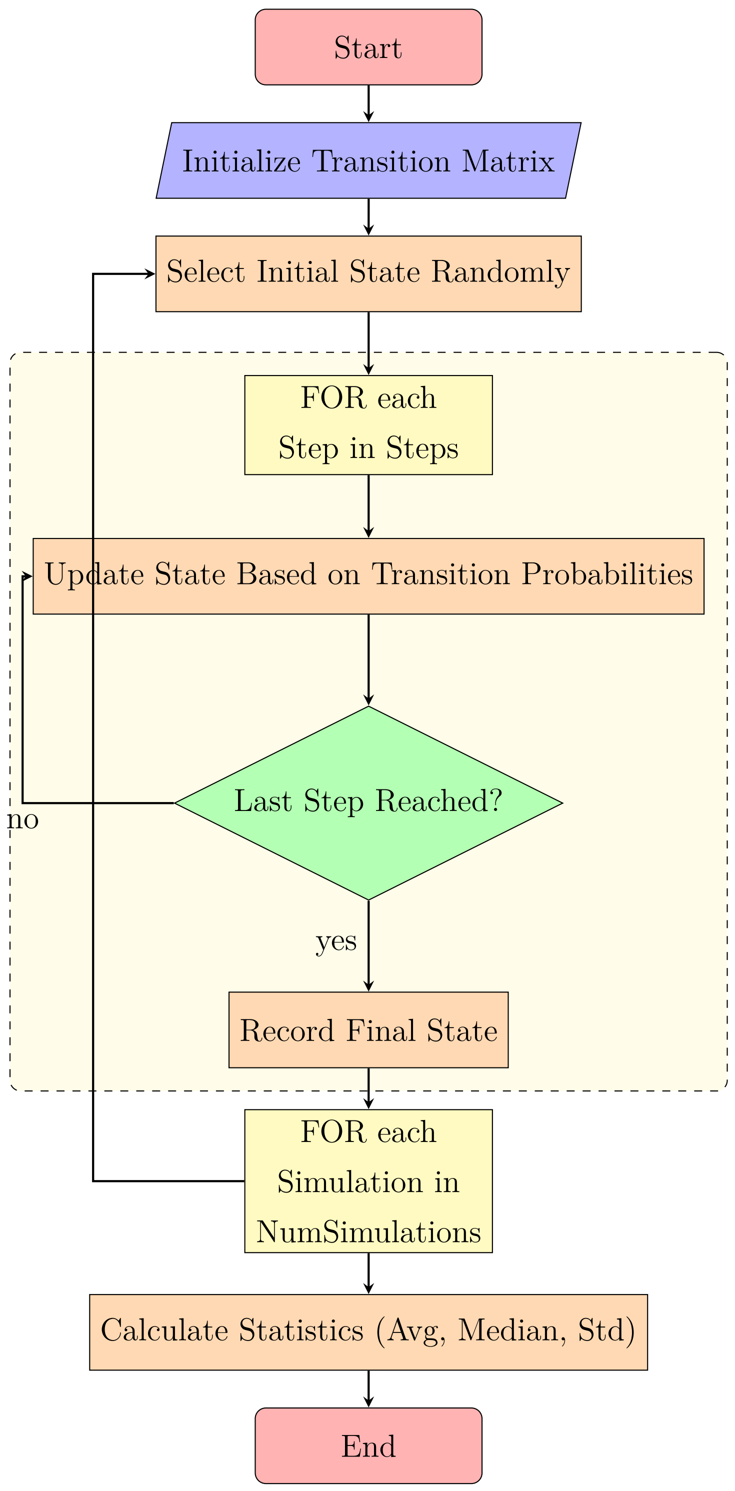
Comparative Studies Using Markov Chain Analysis: You can analyze the stages of regeneration, and the probability of going through each of these stages and also the factors that influence these probabilities, using Markov chains for the regenerative processes of each subject. This research is more than anything to be able to see the significant difference that exists between the regeneration between one and another.(Bondor and Drugan 2011).

For instance, research employing the CRISPR-Cas9 technology has been utilized to pinpoint and modify genes essential to regeneration, providing understanding of the phenotypic consequences of certain genetic modifications. These studies contribute significantly to our knowledge of axolotl regeneration and offer useful data for using Markov chain models. "Dr. Voss and his team are using the CRISPR-Cas9 technique to demonstrate the phenotypic effect of removing certain genes. CRISPR-Cas9 is derived from a bacterial self-defense system"(NIH 2021).

But not everything is as incredible and easy as it seems, although the axolotl is very outstanding in its regenerative ability, it has a very long genome (32 gb) 10 times larger than that of the human approximately, which has been a major barrier to genetic analysis, even so biology is advancing and has already covered a good part of it, but it is still a complication in the study of regeneration(Smith et al. 2019).

4 Markov chain-pseudocode and algorithm

4.1 Flowchart


Start
Initialize Transition Matrix
Select Initial State Randomly
FOR each Step in Steps
Update State Based on Transition Probabilities
Last Step Reached?
no
yes
Record Final State
FOR each Simulation in NumSimulations
Calculate Statistics (Avg, Median, Std)
End

4.2 Flowchart Explanation

1. **Initialization of the transition matrix:** the transition matrix is used to start the regeneration process, this matrix contains the probabilities of changing from one state to another during this process. The matrix contains 4 different states from no regeneration to complete regeneration.
2. **State selection:** you can choose a state or set a random one, this will depend on the user.
3. **FOR each step in steps:** this FOR loop simulates the regeneration process over a set number of steps, each step represents a period of time in the regeneration process.
4. **State change:** during each iteration in the FOR loop, the state changes depending on the probabilities of the transition matrix.
5. **Last state reached?:** at this point the algorithm checks if the last step of the loop has been reached. If not, the loop continues, if yes, the loop ends.
6. **Save the last state:** once the loop is finished the final state of regeneration is saved.
7. **for each simulation in NumSimulations:** this is another loop that serves for the simulation of monte carlo. It repeats the whole regeneration process a certain number of times in order to make a statistical analysis.
8. **Calculate statistics:** after all the Monte Carlo simulations have been completed, the average, median and standard deviation are calculated.

4.3 Pseudocode

```
ALGORITHM Regeneration Simulation
    BEGIN
        # Step 1: Start
        Create the transition matrices for axolotls using parts of his body like Tail,
        Spinal Cord,Internal Organs, and Skin)
        Do the same but for humans (Scratch, Finger Tip, Arm, Liver)

        #Step 2: Initialize the transition matrix for the
        regeneration process with the 4 states "no regeneration",
        "start of regeneration", "partial regeneration", "complete regeneration".
        #Step 3: Assign a number of times you want the
        regeneration process to be repeated.
                Steps = ?.

        # Step 4: FOR i= 0 in NumSimulaton do:
a. Select the initial state
b. As long as the number of states has not been reached:
i. Update the current state based on the assigned transition matrix.
ii. Increment the step by 1.
d. Save the final state.

        # Step 5: repeat the regeneration process a number of
        times assigned for the Monte Carlo simulation and from
        there get the average median and standard deviation.

       PRINT and END
```

5 Coding section

To solve this problem and to find the regenerative capacities of each species, multiple transition matrices were used to compare the various body parts of each species:

```
# Transition matrix for axolotl tail
transition_axolotl_tail = np.array([[0.05, 0.15, 0.1, 0.7],
                                    [0.0, 0.05, 0.15, 0.8],
                                    [0.0, 0.0, 0.1, 0.9],
                                    [0.0, 0.0, 0.0, 1.0]])

# Transition matrix for axolotl Spinal Cord
transition_axolotl_Spine = np.array([[0.1, 0.2, 0.4, 0.3],
                                     [0.0, 0.1, 0.3, 0.6],
                                     [0.0, 0.0, 0.2, 0.8],
                                     [0.0, 0.0, 0.0, 1.0]])

#Transition matrix for axolotl internal organs
transition_axolotl_organs = np.array([[0.2, 0.3, 0.3, 0.2],
                                      [0.0, 0.2, 0.4, 0.4],
                                      [0.0, 0.0, 0.3, 0.7],
                                      [0.0, 0.0, 0.0, 1.0]])

#Transition matrix for axolotl Skin
transition_axolotl_skin = np.array([[0.05, 0.15, 0.1, 0.7],
                                    [0.0, 0.05, 0.15, 0.8],
                                    [0.0, 0.0, 0.1, 0.9],
                                    [0.0, 0.0, 0.0, 1.0]])
```

Listing 1: Python code for transition matrices of the axolotl

these transition matrices are those of the axolotl, each one represents a different part of the axolotl's body, each transition matrix is different since it will express a different regeneration process. We will also have the matrices of the human so that we can compare each species, these are those of the human:

```

#Transition matrix for Human Scrape
transition_human_scrape = np.array([[0.1, 0.3, 0.4, 0.2],
                                    [0.0, 0.1, 0.4, 0.5],
                                    [0.0, 0.0, 0.3, 0.7],
                                    [0.0, 0.0, 0.0, 1.0]])

#Transition matrix for Human Fingertip
transition_human_fingertip = np.array([[0.3, 0.3, 0.3, 0.1],
                                       [0.0, 0.3, 0.4, 0.3],
                                       [0.0, 0.0, 0.4, 0.6],
                                       [0.0, 0.0, 0.0, 1.0]])
#Transition matrix for Human Arm
transition_human_arm = np.array([[0.9, 0.09, 0.01, 0.0],
                                 [0.1, 0.8,  0.1,  0.0],
                                 [0.0, 0.0,  1.0,  0.0],
                                 [0.0, 0.0,  1.0,  0.0]])

#Transition matrix for Human Liver
transition_human_liver = np.array([[0.3, 0.3, 0.3, 0.1],
                                   [0.0, 0.3, 0.4, 0.3],
                                   [0.0, 0.0, 0.4, 0.6],
                                   [0.0, 0.0, 0.0, 1.0]])
```

Listing 2: Python code for transition matrices of the Human

All these transition matrices have some states to define the regeneration process in which each part of the body is, the states are four and are as follows:

```python

states = ["No regeneration", "Initiation of regeneration", "Partial regeneration", "Complete regeneration"]
```

Listing 3: States for the regeneration process

A function is used to simulate the entire regenerative process which will be applied to each of the matrices shown above.

```python
def simulate_regeneration_process(transition_matrix, steps):

    #actual_state = random.choice(range(len(transition_matrix)))
    actual_state = 0


    sequence_states = [actual_state]


    for _ in range(steps - 1):

        actual_state = np.random.choice(range(len(transition_matrix)), p=transition_matrix[actual_state])
        sequence_states.append(actual_state)

    return sequence_states
```

Listing 4: Function for the regeneration process

This function receives two parameters to be able to work, in the code it can be observed that it needs of a transition matrix and a number of steps to work, the state is assigned in 0 that is to say in initial state or of no regeneration this with the purpose of comparing both species in the worst state. There is also a list called **state_sequence** that will help to save all the states that come out of this regenerative process.

- There is a **FOR** loop that runs with the specified number of steps with one step less since the first state has already been selected.

- Inside this loop we will assign the **current_state** with the function **np.random.choice**, this function receives two parameters that are **range(len(transition_matrix))** and **p=transition_matrix**[**current_state**].
- The first parameter will generate a sequence from 0 to 3, which will represent a state in the regeneration process, and the second parameter will set the probabilities in the function depending on the current state.
- Then the current status will be updated and saved in the list previously created using **.append**.
- At the end of the whole process **return state_sequence** will be used to return the list containing all the states visited during the simulation.

to be able to print the data that the function will give us we will use the following format:

```python
steps_simulations=10
example_simulation_tail = simulate_regeneration_process(
    ↪ transition_axolotl_tail, steps_simulations)

print("Regeneration␣Processes␣of␣Human␣and␣Axolotl\n␣Hector␣David␣Peralta
    ↪ ␣Ramirez")

print("Tail␣-␣Axolotl,␣Number␣of␣Steps:␣" + str(10))
print(example_simulation_tail)
```

Listing 5: Print the results of the function

Output:

```
Regeneration Processes of Human and Axolotl
 Hector David Peralta Ramirez
Tail - Axolotl, Number of Steps: 10
[0, 1, 3, 3, 3, 3, 3, 3, 3, 3]
```

Listing 6: The results of the function

In the function simply assigned a number of **steps**, in this case **10**, and assigned the transition **matrix = transition_axolotl_tail**, all this assigned to a variable called **example_simulation_tail**, then using print assigned the project name, student name, name of the body part specifying the species and number of steps. The results can be seen in the output.

5.1 Monte Carlo - Coding Section

The Monte Carlo simulation is used in the code to analyze the average, median and standard deviation, to get a broader understanding of how the whole model behaves under different conditions and how varied these results can be after several iterations.

```

def simulation_monte_carlo(transition_matrix, steps, num_simulations):
    final_results = []

    for _ in range(num_simulations):
        sequence = simulate_regeneration_process(transition_matrix, steps
            ↪ )
        final_state = sequence[-1]
        final_results.append(final_state)

    return final_results
```

Listing 7: Monte Carlo base function

Function Explanation:

- **def simulation_monte_carlo(transition_matrix, steps, num_simulations):** The function uses 3 parameters which are the transition matrix, a number of steps and a number of simulations, this function is similar to the regeneration process but adds one more parameter which is the **Num_simulations**, which will be the independent number of simulations that will be done.
- **Final_results:** In this variable a list is created to store the final state of each simulation.
- **for _ in range(num_simulations):** in this **FOR** loop iterates the number of times assigned by the **num_simulations** variable, running a different simulation in each iteration.
- The other steps are similar to the regeneration process since the same function is used.

Two FOR loops and a list were used to print all the information from the Monte Carlo simulation and to analyze all the data from the multiple simulations run.

```
matrices = [
    transition_axolotl_tail, transition_axolotl_Spine,
    transition_axolotl_organs, transition_axolotl_skin,
    transition_human_scrape, transition_human_fingertip,
    transition_human_arm, transition_human_liver
]
```

Listing 8: Matrices List

```
num_simulations = 1000
steps = 10


results_all_matrices = {}
for i, matrix in enumerate(matrices):
    results = simulation_monte_carlo(matrix, steps, num_simulations)
    results_all_matrices[f"matrix␣{i+1}"] = results

for i, (name, results) in enumerate(results_all_matrices.items()):
    average = np.mean(results)
    median = np.median(results)
    standard_deviation = np.std(results)
    print(f"Results␣for␣{name}:")
    print(f"Average␣of␣the␣final␣state:␣{average:.2f}")
    print(f"Median␣of␣the␣final␣state:␣{median}")
    print(f"Standard␣deviation␣of␣the␣final␣state:␣{standard_deviation:.2
        ↪ f}\n")
```

Listing 9: Monte Carlo Print

- A data structure called dictionary is created to store the results of each Monte Carlo simulation.
- **for i, matrix in enumerate(matrices):**this loop will iterate over each of the transition matrices stored in the previous list.
- **results = simulation_monte_carlo(matrix, steps, num_simulations):**With the set number of steps and simulations the Monte Carlo function is executed and stored in the results variable.
- **results_all_matrices[f"matrix i+1"]**:Here the results of the simulations are stored in the dictionary that we created at the beginning, the iterator is used since this type of data structure uses a key to identify the matrix.
- **for i, (name, results) in enumerate(results_all_matrices.items()):**this function iterates on the results obtained by the Monte Carlo simulation.

- For each result, the numpy library is used to obtain the average, median, and standard deviation. In the code the abbreviation np is used to express the use of this library, then the results are printed.

```

Results for matrix 1:
Average of the final state: 3.00
Median of the final state: 3.0
Standard deviation of the final state: 0.00
```

Listing 10: Output of Monte Carlo simulation

5.2 Stationary Distribution

```

def calculate_stationary_distribution(matrix, n_steps):
    result_matrix = np.copy(matrix)
    for _ in range(n_steps - 1):
        result_matrix = np.dot(result_matrix, matrix)
    return result_matrix
```

Listing 11: Stationary Distribution Code

Function Explanation

- def calculate_stationary_distribution(matrix, n_steps):the function takes two parameters which are the selected transition matrix and a certain number of steps, the number of steps will mean the number of times this matrix will be multiplied by itself.
- A variable called **result_matrix** will be used to store a copy of the transition matrix, so as not to lose it during multiplications.
- The second block will simply repeat the process of multiplying the matrix by itself, subtracting one since we have an initial copy.
- For the multiplications we will use the numpy library using the **np.dot** function.

5.3 Math Behind Stationary Distribution

To understand a little more about how to draw the stationary distribution we will do the first 3 steps simply to see how the axolotl-tail matrix works, as follows

:

$$P = \begin{bmatrix} 0.05 & 0.15 & 0.1 & 0.7 \\ 0.0 & 0.05 & 0.15 & 0.8 \\ 0.0 & 0.0 & 0.1 & 0.9 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

$$P^2 = \begin{bmatrix} 0.05 & 0.15 & 0.1 & 0.7 \\ 0.0 & 0.05 & 0.15 & 0.8 \\ 0.0 & 0.0 & 0.1 & 0.9 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \times \begin{bmatrix} 0.05 & 0.15 & 0.1 & 0.7 \\ 0.0 & 0.05 & 0.15 & 0.8 \\ 0.0 & 0.0 & 0.1 & 0.9 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

$$\textbf{Result:} \quad P^2 = \begin{bmatrix} 0.0025 & 0.015 & 0.0375 & 0.945 \\ 0.0 & 0.0025 & 0.0225 & 0.975 \\ 0.0 & 0.0 & 0.01 & 0.99 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

to continue calculating the following steps we will simply multiply the matrix by itself and as we advance we will continue multiplying by the original one the result that it gives us.

$$P^3 = \begin{bmatrix} 0.0025 & 0.015 & 0.0375 & 0.945 \\ 0.0 & 0.0025 & 0.0225 & 0.975 \\ 0.0 & 0.0 & 0.01 & 0.99 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \times \begin{bmatrix} 0.05 & 0.15 & 0.1 & 0.7 \\ 0.0 & 0.05 & 0.15 & 0.8 \\ 0.0 & 0.0 & 0.1 & 0.9 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

$$\textbf{Result:} \quad P^3 = \begin{bmatrix} 1.2500e-04 & 1.1250e-03 & 6.2500e-03 & 9.9250e-01 \\ 0.0000e+00 & 1.2500e-04 & 2.6250e-03 & 9.9725e-01 \\ 0.0000e+00 & 0.0000e+00 & 1.0000e-03 & 9.9900e-01 \\ 0.0000e+00 & 0.0000e+00 & 0.0000e+00 & 1.0000e+00 \end{bmatrix}$$

5.4 Stationary Distribution - Output

As can be observed, most of the matrices will tend to a state of maximum regeneration, especially those of the axolotl due to its high regenerative capacity, two outputs of the stationary distribution will be shown, one of the human arm and the other of the axolotl tail.

```

Stationary distribution for human_arm:
[[7.4e-323 3.5e-323 1.0e+000 0.0e+000]
```

```
 [7.4e-323 3.5e-323 1.0e+000 0.0e+000]
 [0.0e+000 0.0e+000 1.0e+000 0.0e+000]
 [0.0e+000 0.0e+000 1.0e+000 0.0e+000]]
```

Listing 12: Output Stationary Distribution - Human Arm

```

Stationary distribution for axolotl_tail:
[[0. 0. 0. 1.]
 [0. 0. 0. 1.]
 [0. 0. 0. 1.]
 [0. 0. 0. 1.]]
```

Listing 13: Output Stationary Distribution - Axolotl Tail

The axolotl has no problem in regenerating its tail from its body as you can see, but the human is unable to regenerate its arm, which demonstrates the great capacity and ability of the axolotl.

6 Full Code

```python

import numpy as np
import matplotlib.pyplot as plt
import random

#Hector David Peralta Ramirez
#Prague City University
#Programming Fundamentals


# States for the regeneration process
states = ["No regeneration", "Initiation of regeneration", "Partial 
    ↪ regeneration", "Complete regeneration"]

# Transition matrices for the axolotl (Tail, Spinal Cord, Internal organs
    ↪ , Skin)
transition_axolotl_tail = np.array([[0.05, 0.15, 0.1, 0.7],
                                    [0.0, 0.05, 0.15, 0.8],
                                    [0.0, 0.0, 0.1, 0.9],
                                   [0.0, 0.0, 0.0, 1.0]])

transition_axolotl_Spine = np.array([[0.1, 0.2, 0.4, 0.3],
                                     [0.0, 0.1, 0.3, 0.6],
                                     [0.0, 0.0, 0.2, 0.8],
                                     [0.0, 0.0, 0.0, 1.0]])

transition_axolotl_organs = np.array([[0.2, 0.3, 0.3, 0.2],
                                     [0.0, 0.2, 0.4, 0.4],
                                     [0.0, 0.0, 0.3, 0.7],
                                     [0.0, 0.0, 0.0, 1.0]])

transition_axolotl_skin = np.array([[0.05, 0.15, 0.1, 0.7],
                                   [0.0, 0.05, 0.15, 0.8],
                                   [0.0, 0.0, 0.1, 0.9],
                                   [0.0, 0.0, 0.0, 1.0]])

# Transition matrices for the Humans (Scrape, fingertip, arm , liver)
transition_human_scrape = np.array([[0.1, 0.3, 0.4, 0.2],
                                   [0.0, 0.1, 0.4, 0.5],
                                   [0.0, 0.0, 0.3, 0.7],
                                   [0.0, 0.0, 0.0, 1.0]])

transition_human_fingertip = np.array([[0.3, 0.3, 0.3, 0.1],
                                      [0.0, 0.3, 0.4, 0.3],
                                      [0.0, 0.0, 0.4, 0.6],
                                      [0.0, 0.0, 0.0, 1.0]])

transition_human_arm = np.array([[0.9, 0.09, 0.01, 0.0],
                                [0.1, 0.8,  0.1,  0.0],
                                [0.0, 0.0,  1.0,  0.0],
                                [0.0, 0.0,  1.0,  0.0]])

transition_human_liver = np.array([[0.3, 0.3, 0.3, 0.1],
                                  [0.0, 0.3, 0.4, 0.3],
                                  [0.0, 0.0, 0.4, 0.6],
                                  [0.0, 0.0, 0.0, 1.0]])

```

```python
   #Function to simulate the regeneration process
def simulate_regeneration_process(transition_matrix, steps):
    # Select an starting state, 0 = No regeneration
    actual_state = 0

    # Use a list to save the states that are going to be used in the process
    #Of Regeneration
    sequence_states = [actual_state]

    # Do the Simulation for the number of steps - 1, because we already have one
    for _ in range(steps - 1):
    #Select the next step based on the probabilities of each transition matrix.
        actual_state = np.random.choice(range(len(transition_matrix)), p=transition_matrix[actual_state])
        sequence_states.append(actual_state)

    return sequence_states

steps_simulations=10
#Examples of simulations for each transition matrix
example_simulation_tail = simulate_regeneration_process(transition_axolotl_tail, steps_simulations)
example_simulation_spine = simulate_regeneration_process(transition_axolotl_Spine, steps_simulations)
example_simulation_organs = simulate_regeneration_process(transition_axolotl_organs, steps_simulations)
example_simulation_skin = simulate_regeneration_process(transition_axolotl_skin, steps_simulations)

example_simulation_scrape = simulate_regeneration_process(transition_human_scrape, steps_simulations)
example_simulation_fingertip = simulate_regeneration_process(transition_human_fingertip, steps_simulations)
example_simulation_arm = simulate_regeneration_process(transition_human_arm, steps_simulations)
```

```python
example_simulation_liver = simulate_regeneration_process(transition_human_liver, steps_simulations)


#Print the results of each transition matrix, also including the name of the project and the student name.
print("Regeneration Processes of Human and Axolotl\n Hector David Peralta Ramirez")

print("Tail - Axolotl, Number of Steps: " + str(10))
print(example_simulation_tail)
print("Spinal Cord - Axolotl, Number of Steps: " + str(10))
print(example_simulation_spine)
print("Organs - Axolotl, Number of Steps: " + str(10))
print(example_simulation_organs)
print("Skin - Axolotl, Number of Steps: " + str(10))
print(example_simulation_skin)

print("Scrape - Human, Number of Steps" + str(10))
print(example_simulation_scrape)
print("FingerTip - Human, Number of Steps" + str(10))
print(example_simulation_fingertip)
print("Arm - Human, Number of Steps" + str(10))
print(example_simulation_arm)
print("Liver - Human, Number of Steps" + str(10))
print(example_simulation_liver)



def simulation_monte_carlo(transition_matrix, steps, num_simulations):
    #Initialize a list for all the results of the simulation
    final_results = []

    #Use for loop to repeat the regeneration process multiple times for independent simulations
    for _ in range(num_simulations):
        #Use the specified transition matrix with certain number of steps
        sequence = simulate_regeneration_process(transition_matrix, steps
            ↪ )
        final_state = sequence[-1]
        #add the results to the lists created before
        final_results.append(final_state)

    return final_results

#Create a list of matrices to apply monte carlo simulation
matrices = [
    transition_axolotl_tail, transition_axolotl_Spine,
    transition_axolotl_organs, transition_axolotl_skin,
    transition_human_scrape, transition_human_fingertip,
    transition_human_arm, transition_human_liver
]

#set number of simulations and steps
num_simulations = 10000
steps = 10

#Create a dictionary to save all the matrices
results_all_matrices = {}
#use a loop for each matrix on the list "Matrices"
for i, matrix in enumerate(matrices):
    #Run monte carlo simulation
    results = simulation_monte_carlo(matrix, steps, num_simulations)
    #Results will be added to the dictionary with their respective number
    results_all_matrices[f"matrix␣{i+1}"] = results

#Print the results of the simulation
for i, (name, results) in enumerate(results_all_matrices.items()):
    #use the library numpy to calculate the Average, median and standard
        ↪ deviation.
    average = np.mean(results)
    median = np.median(results)
    standard_deviation = np.std(results)
    #Print the results.
    print(f"Results␣for␣{name}:")
    print(f"Average of the final state: {average:.2f}")
    print(f"Median of the final state: {median}")
    print(f"Standard deviation of the final state: {standard_deviation:.2f}\n")




# Menu for stationary distribution, User can select which matrix will be chosen for the
#stationary distribution.

matrices1 = {
    #all matrices being declared again with a respective number for the menu (Similar to a Switch)

    1: ("axolotl_tail", np.array([[0.05, 0.15, 0.1, 0.7],
                                  [0.0, 0.05, 0.15, 0.8],
                                  [0.0, 0.0, 0.1, 0.9],
                                  [0.0, 0.0, 0.0, 1.0]])),
    2: ("axolotl_spine", np.array([[0.1, 0.2, 0.4, 0.3],
                                   [0.0, 0.1, 0.3, 0.6],
                                   [0.0, 0.0, 0.2, 0.8],
                                   [0.0, 0.0, 0.0, 1.0]])),
    3: ("axolotl_organs", np.array([[0.2, 0.3, 0.3, 0.2],
                                    [0.0, 0.2, 0.4, 0.4],
                                    [0.0, 0.0, 0.3, 0.7],
                                    [0.0, 0.0, 0.0, 1.0]])),
    4: ("axolotl_skin", np.array([[0.05, 0.15, 0.1, 0.7],
                                    [0.0, 0.05, 0.15, 0.8],
                                    [0.0, 0.0, 0.1, 0.9],
                                    [0.0, 0.0, 0.0, 1.0]])),
    5: ("human_scrape", np.array([[0.1, 0.3, 0.4, 0.2],
                                [0.0, 0.1, 0.4, 0.5],
                                [0.0, 0.0, 0.3, 0.7],
                                [0.0, 0.0, 0.0, 1.0]])),
    6: ("human_fingertip", np.array([[0.3, 0.3, 0.3, 0.1],
                                   [0.0, 0.3, 0.4, 0.3],
                                          [0.0, 0.0, 0.4, 0.6],
                                          [0.0, 0.0, 0.0, 1.0]])),
    7: ("human_arm", np.array([[0.9,  0.09,  0.01,  0.0],
                                [0.1,  0.8,   0.1,   0.0],
                                [0.0,  0.0,   1.0,   0.0],
                                [0.0,  0.0,   1.0,   0.0]])),

    8: ("human_liver", np.array([[0.9, 0.09, 0.01, 0.0],
                                [0.1, 0.8, 0.1, 0.0],
                                [0.2, 0.1, 0.7, 0.0],
                                [0.0, 0.0, 0.0, 1.0]]))

}

#Function to calculate de stationary distribution with two parameters
def calculate_stationary_distribution(matrix, n_steps):
    #Store the original matrix
    result_matrix = np.copy(matrix)
    #Using a loop to multipy the matrix multiple times using numpy with the .dot function.
    for _ in range(n_steps - 1):
        result_matrix = np.dot(result_matrix, matrix)
    return result_matrix

# Menu for the User
print("Please␣choose␣a␣matrix:")
for number, (name, _) in matrices1.items():
    print(f"{number}:␣{name}")

# Ask to the user to choose a matrix
user_choice = int(input("Enter␣the␣number␣of␣your␣choice:␣"))

if user_choice in matrices1:
    _, chosen_matrix = matrices1[user_choice]
    n_steps = 10000000  # 10 mill steps
    stationary_distribution = calculate_stationary_distribution(chosen_matrix, n_steps)
    print(f"Stationary␣distribution␣for␣{matrices1[user_choice][0]}:\n{
        ↪ stationary_distribution}")
else:
    print("Invalid␣choice.")
```

Listing 14: Full Code

7 Conclusion

This project is very interesting, to be able to see how markov chains together with transition matrices can simulate the regeneration of different species is fascinating. The use of the axolotl was an important part of the project because giving importance to this type of animals that maybe are not so well known but have impressive capabilities that could inspire future research and the fact of analyzing how superior they are to humans in regeneration makes you value these animals even more. Although the use of stationary distribution teaches the capacity of both species, the use of Monte Carlo helps to analyze it in another way since the stationary distribution can also be carried away by absorbing states, the project was made with the intention of looking for a different approach to Markov chains and draw the reader's attention with fascinating animals. (Echeverri 2020)

Figure 1: Ambystoma mexicanum. The source of the image is axolotlowner 2024.

References

axolotlowner (2024). *Speckled Leucistic Axolotl Front-facing Full Body View on White Background.* Shutterstock. URL: `https://www.shutterstock.com/es/image-photo/speckled-leucistic-axolotl-frontfacing-full-body-2156757677` (visited on 01/17/2024).

Bondor, C. and T. Drugan (2011). “Statistical Methods in Bioinformatics: Markov Chains”. In: *Applied Medical Informatics* 11, pp. 47–53. (Visited on 12/20/2023).

Catherine McCusker, David M Gardiner (2021). *The axolotl model for regeneration and aging research: a mini-review.* Ed. by NIH. DOI: `10.1159/000323761`. URL: `https://pubmed.ncbi.nlm.nih.gov/21372551/` (visited on 12/01/2023).

Echeverri, K. (2020). “The various routes to functional regeneration in the central nervous system”. In: *Communications Biology* 3. DOI: `10.1038/s42003-020-0773-z`. (Visited on 12/20/2023).

Henry Maltby Worranat Pakornrat, Jeremy Jackson (2023). *Markov Chains.* Ed. by Brilliant. URL: `https://brilliant.org/wiki/markov%20chains/#:~:text=A%20Markov%20chain%20is%20a,possible%20future%20states%20are%20fixed` (visited on 12/01/2023).

Mohite, Priti, Sarita Wadkar, and Pooja Zanjurne (Jan. 2023). “Study of Stock Market using Markov Chains”. In: URL: `https://www.researchgate.net/publication/366986997_Study_of_Stock_Market_using_Markov_Chains` (visited on 11/11/2023).

NIH (2021). *The Amazing Axolotl: A Valuable Model for Regenerative Medicine.* Ed. by NIH. URL: `https://orip.nih.gov/about-orip/research-highlights/amazing-axolotl-valuable-model-regenerative-medicine` (visited on 12/01/2023).

Norris, James (Oct. 2011). “Markov Chains”. In: ed. by University of Cambridge, p. 56. (Visited on 12/02/2023).

Rupinder Sekhon, Roberta Bloom (July 2022). *Introduction to Markov Chains.* Ed. by Anza College. URL: `https://math.libretexts.org/Bookshelves/Applied_Mathematics/Applied_Finite_Mathematics_(Sekhon_and_Bloom)/10%3A_Markov_Chains/10.01%3A_Introduction_to_Markov_Chains` (visited on 12/01/2023).

Smith, J., Nataliya Y. Timoshevskaya, Vladimir A. Timoshevskiy, Melissa C. Keinath, D. Hardy, and S. Voss (2019). “A chromosome-scale assembly of the axolotl genome”. In: *Genome Research* 29, pp. 317–324. DOI: `10.1101/gr.241901.118`. (Visited on 12/20/2023).