ECE 4110/5110

Random Signals in Communication and Signal Processing

Homework 4

Name:	Yiyuan (Evan) Lin
Cornell ID:	5515425
NetID:	yl3663
Major:	Electrical and Computer Engineering (Ph.D.)
Course Professor:	Dr. Kevin Tang

Oct 24, 2023

Contents

1 Problem 1 and Corresponding Solution

P1. Branching Process

In this problem, we will simulate a branching process. Let Z_0, Z_1, ... be a branching process where the number of children of each node comes from a Poisson distribution with parameter λ.

(a) First assume $\lambda = 2$, simulate this branching process for 5 generations (until Z_5) and record the number of nodes (Z_5). Repeat this simulation 20 times and compute the average number of children at 5th generation. Does your result match with the theoretical result?

(b) Now, assume $\lambda = \frac{1}{2}$, simulate this branching process until the branching process is extinct. Repeat this simulation 20 times and record their extinction times. How many times was the process extinct after 1 generation? Does that match the theoretical results? What is the average time to extinction?

Solution:

(a) We know $\lambda = 2$ and $n = 5$, let $\mu = E(Z_1) = \lambda = 2$. Then $E(Z_n) = \mu^n = \lambda^n$

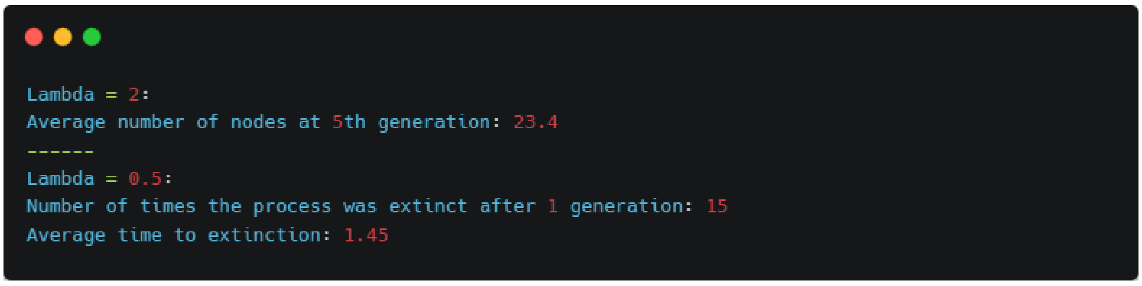
Hence, the theoretical result of Z_5 is

$$Z_5 = \lambda^n = 32$$

(b) To find the probability that the branching process is extinct after 1 generation, we can just set $k = 0$ to the PMF of Poison Distribution $p_x(k) = \frac{e^{-\lambda}\lambda^k}{k!}$

$$P(Z_1 = 0) = p_x(0) = \frac{e^{-\lambda}\lambda^0}{0!} = e^{-0.5} = 0.6065306597126334$$

For the simulation, we first repeat the simulation of this branching process 20 times. The result is shown below:



```
Lambda = 2:
Average number of nodes at 5th generation: 23.4
------
Lambda = 0.5:
Number of times the process was extinct after 1 generation: 15
Average time to extinction: 1.45
```

When $\lambda = 2$,

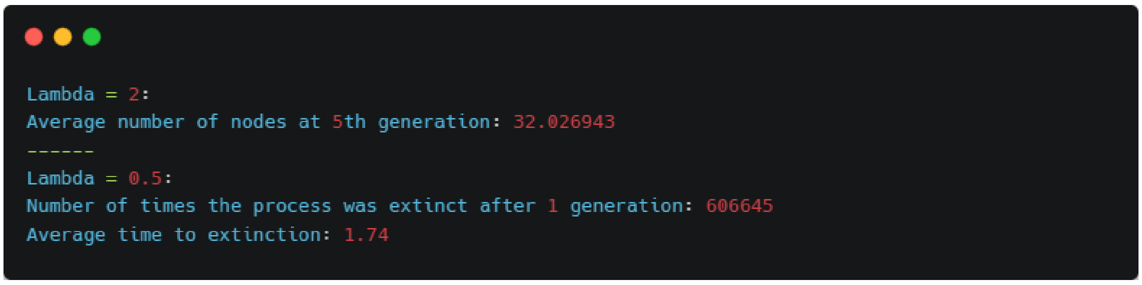
- one of the average number of nodes at 5th generation is 23.4.

When $\lambda = \frac{1}{2}$,

- the process is extinct after 1 generation for 15 times in 20 simulations.
- the average time to extinction is 1.45.

The simulation results don't look so much like the theoretical values.

Since the repetition 20 is not large enough, it is hard to conclude and verify the theoretical result. Then let's repeat this simulation 1000000 times and the result is shown below.



```
Lambda = 2:
Average number of nodes at 5th generation: 32.026943
------
Lambda = 0.5:
Number of times the process was extinct after 1 generation: 606645
Average time to extinction: 1.74
```

When $\lambda = 2$,

- one of the average number of nodes at 5th generation is 32.026943.

When $\lambda = \frac{1}{2}$,

- the process is extinct after 1 generation for 606645 times in 100000 simulations.
- the average time to extinction is 1.74.

These match the theoretical results.

2 Problem 2 and Corresponding Solution

P2. Branching Process

Consider a branching process whose family sizes have the geometric mass function $f(k) = qp^k$, $k \geq 0$, where $p + q = 1$, and let Zn be the size of the nth generation. Let $T = min\{n : Zn = 0\}$ be the extinction time, and suppose that $Z_0 = 1$. Find $P(T = n)$.

Solution:

The number Z_n of n-th generation descendants satisfies

$$P(Z_n = 0) = G_n(0) = \begin{cases} \frac{n}{n+1} & \text{if } p = q \\ \frac{q(p^n - q^n)}{p^{n+1} - q^{n+1}} & \text{if } p \neq q \end{cases}$$

Hence, for $n \geq 1$,

$$P(T = n) = P(Z_n = 0) - P(Z_{n-1} = 0) = \begin{cases} \frac{1}{n(n+1)} & \text{if } p = q \\ \frac{p^{n-1} q^n (p-q)^2}{(p^n - q^n)(p^{n+1} - q^{n+1})} & \text{if } p \neq q \end{cases}$$

3 Problem 3 and Corresponding Solution

P3. Monte Carlo Methods

We are trying to approximate the number π. One way of doing so is using Monte Carlo Methods and Law of Large Numbers. We will choose two random numbers (x and y) from a uniform distribution U(-1,1). This way we have a random point in the square $[-1, 1] \times [-1, 1]$. Then, we will check whether that point is inside the unit circle.

(a) What is the probability of a random point chosen in the manner above being in the unit circle.

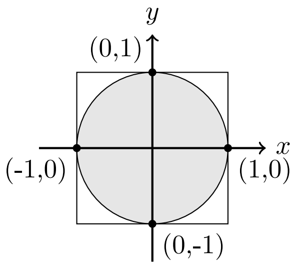
(b) Propose a way to use this fact to estimate π

(c) Write a code to simulate this experiment (at least 1000 samples) and report the results.

Solution:

(a) The desired probability is

$$P(\text{a random point chosen in the manner above being in the unit circle}) = \frac{S_{circle}}{S_{square}} = \frac{\pi}{4}$$

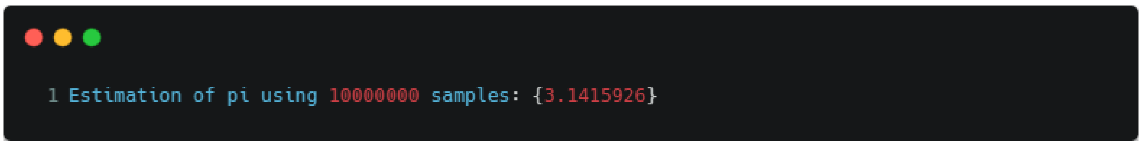


(b) Given the probability P from part (a), we can use Monte Carlo simulation to estimate π. Here's the procedure:

1. Randomly select a large number N of points within the square $[-1, 1] \times [-1, 1]$.
2. Count the number M of these points that fall inside the unit circle.
3. The ratio of the number of points inside the circle to the total number of points is an estimate of the probability P. Thus, $P \approx \frac{M}{N}$.
4. Using the relation $P = \frac{\pi}{4}$, we can estimate π as:

$$\pi \approx 4 \times \frac{M}{N}$$

(c) The result of the simulation is shown below:



```
Estimation of pi using 10000000 samples: {3.1415926}
```

Then we have the estimation of π

$$\pi \approx 3.1415926$$

4 Problem 4 and Corresponding Solution

P4. Metropolis Hastings Algorithm

The Metropolis Hastings algorithm is a beautifully simple algorithm for producing samples from distributions that may otherwise be difficult to sample from.

Suppose we want to sample from a distribution π, which we will call the ”target” distribution. For simplicity we assume that π is a one-dimensional distribution on the real line, although it is easy to extend to more than one dimension (see below).

The MH algorithm works by simulating a Markov Chain, whose stationary distribution is π. This means that, in the long run, the samples from the Markov chain look like the samples from π. As we will see, the algorithm is incredibly simple and flexible. Its main limitation is that, for difficult problems, ”in the long run” may mean after a very long time. However, for simple problems the algorithm can work well.

The transition kernel: To implement the MH algorithm, the user (you!) must provide a ”transition kernel”, Q. A transition kernel is simply a way of moving, randomly, to a new position in space (y say), given a current position (x say). That is, Q is a distribution on y given x, and we will write it $Q(y|x)$.

The MH algorithm: for sampling from a target distribution π, using transition kernel Q, consists of the following steps:

- Initialize, $X_1 = x_1$ say.
- For $t = 1, 2, ...$
  - sample y from $Q(y|x_t)$. Think of y as a ”proposed” value for x_{t+1}.
  - Compute
$$A = min(1, \frac{\pi(y)Q(x_t|y)}{\pi(x_t)Q(y|x_t)})$$
  - A is often called the ”acceptance probability”. with probability A ”accept” the proposed value, and set $x_{t+1} = y$. Otherwise set $x_{t+1} = x_t$.

Example (You will code this part): Assume, we want to sample from the following distribution:

$$\pi(i) = \begin{cases} \frac{1}{2^{6-i}} & \text{for } i = 1, 2, ...5 \\ \frac{1}{2^5} & \text{for } i = 0 \end{cases}$$

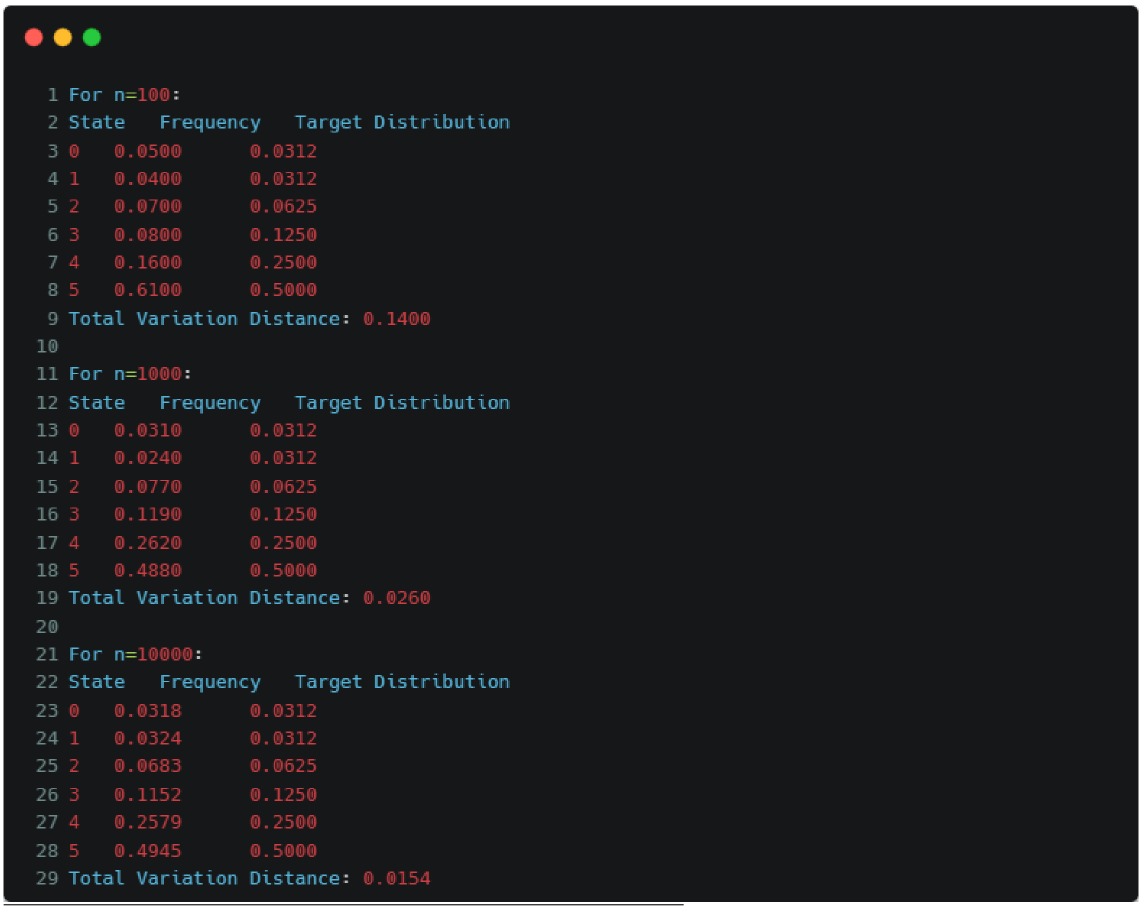
We do that using the following kernel:

$$Q(y|x) = \frac{1}{6}, \qquad \forall y, x \in \{0, 1, 2, 3, 4, 5\}$$

Simulate the above Markov Chain Monte Carlo for $n = 100, 1000, 10000$ steps. Find the frequency of visiting each state and plot the histogram. Compare the results for different n. Does it get closer to π?

Solution:

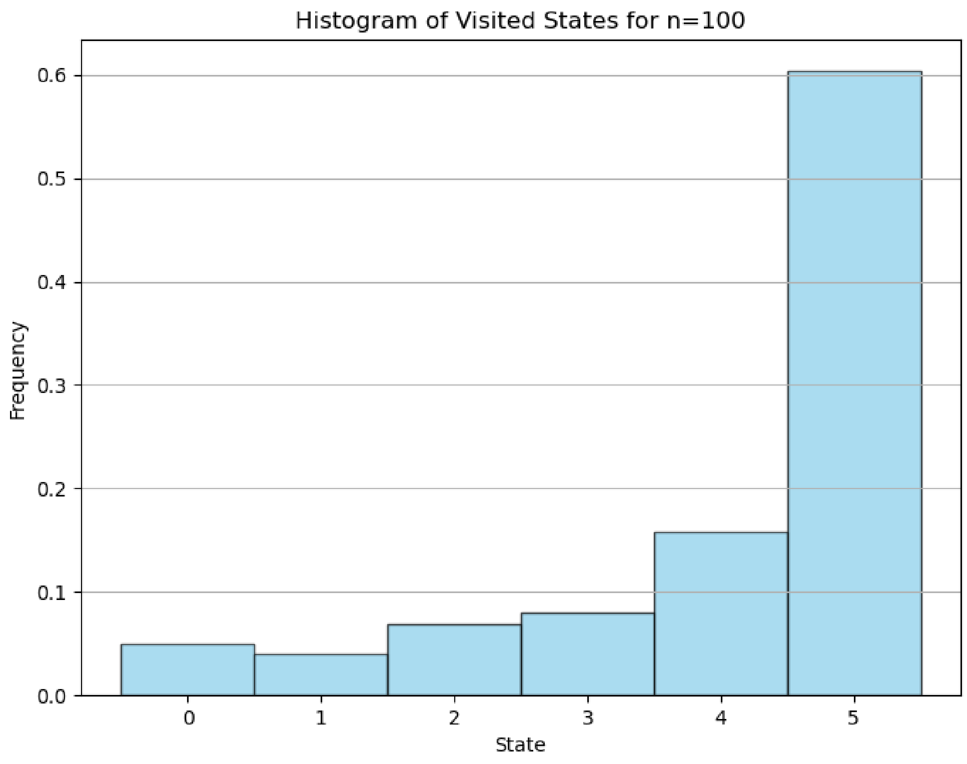
The frequency of visiting each state is shown below.

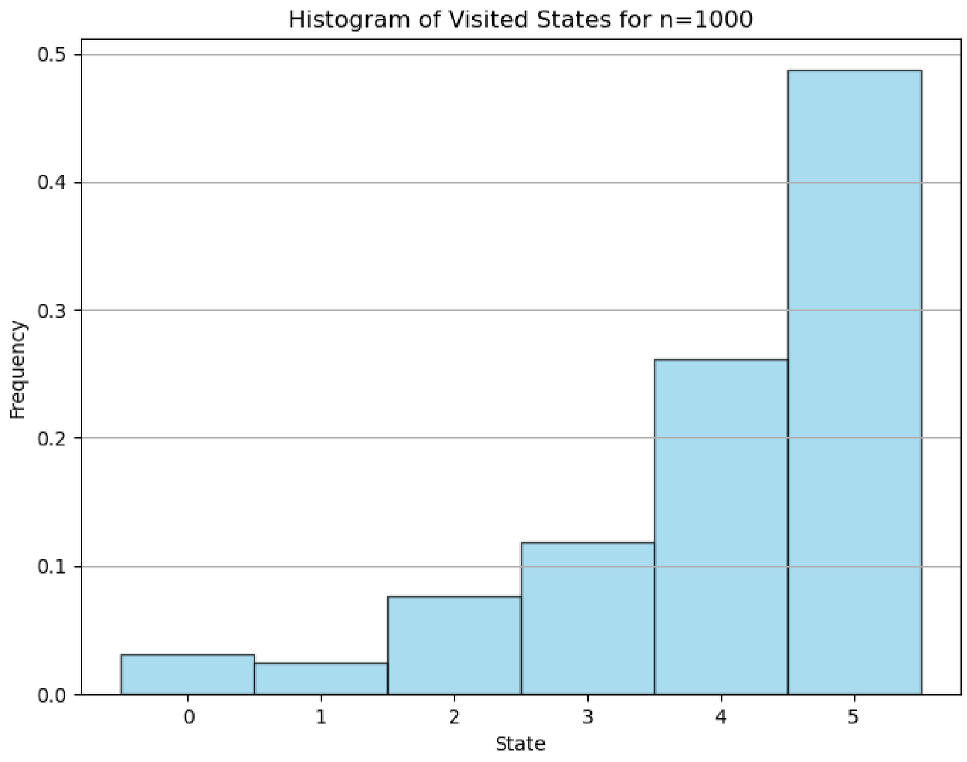


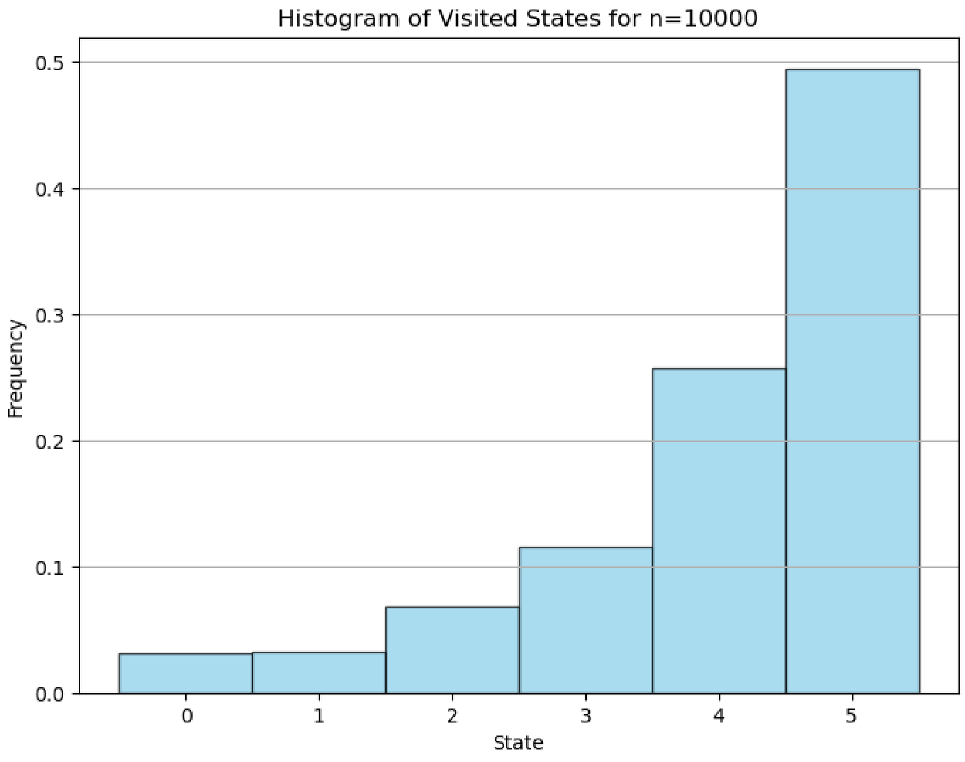
```
For n=100:
State   Frequency   Target Distribution
0   0.0500      0.0312
1   0.0400      0.0312
2   0.0700      0.0625
3   0.0800      0.1250
4   0.1600      0.2500
5   0.6100      0.5000
Total Variation Distance: 0.1400

For n=1000:
State   Frequency   Target Distribution
0   0.0310      0.0312
1   0.0240      0.0312
2   0.0770      0.0625
3   0.1190      0.1250
4   0.2620      0.2500
5   0.4880      0.5000
Total Variation Distance: 0.0260

For n=10000:
State   Frequency   Target Distribution
0   0.0318      0.0312
1   0.0324      0.0312
2   0.0683      0.0625
3   0.1152      0.1250
4   0.2579      0.2500
5   0.4945      0.5000
Total Variation Distance: 0.0154
```

For n=100:	**Total Variation Distance: 0.1400**	
State	Frequency	Target Distribution
0	0.05	0.0312
1	0.04	0.0312
2	0.07	0.0625
3	0.08	0.125
4	0.16	0.25
5	0.61	0.5
For n=1000:	**Total Variation Distance: 0.0260**	
State	Frequency	Target Distribution
0	0.031	0.0312
1	0.024	0.0312
2	0.077	0.0625
3	0.119	0.125
4	0.262	0.25
5	0.488	0.5
For n=10000:	**Total Variation Distance: 0.0154**	
State	Frequency	Target Distribution
0	0.0318	0.0312
1	0.0324	0.0312
2	0.0683	0.0625
3	0.1152	0.125
4	0.2579	0.25
5	0.4945	0.5

The histograms of the frequency of visiting each state for different n are shown below.







One way to evaluate if the simulated frequencies get closer to the target distribution π as n increases is to compute the total variation distance between the simulated frequencies and π. The total variation distance between two probability distributions p and q over a finite state space S is defined as:

$$\mathrm{TV}(p,q) = \frac{1}{2}\sum_{s\in S} |p(s) - q(s)|$$

By calculating and observing the total variation distance for each n, we can find that the distance decreases with increasing n, which indicates that the algorithm is converging to the target distribution.

It is also straightforward to see the histograms and find that the simulated frequencies get closer to the target distribution π as n increases.

5 Problem 5 and Corresponding Solution

P5. Markov Decision Process

A stuntman performs stunts for a living. At each day, she is either "Healthy", has "Minor injuries", or has "Major injuries".

- If she is healthy, She can choose whether to do a low-risk stunt or a high-risk stunt.
  - Low Risk Stunt: This stunt Pays 100 Dollars (reward). She will stay Healthy with probability 0.7, Sustain minor injuries with probability 0.3. (She will never sustain a major injury.)
  - High-Risk Stunt: This stunt pays 400$. But she will sustain major injuries.
- If she has minor injuries she will be healthy the next day with probability 1.
- If she has major injuries she will still have major injuries the day after with probability 0.5. She will heal a little bit and have minor injuries with probability 0.5.

The days she is in hospital/injured, she will make no money. Now, answer the following questions:

(a) Assume, the stunt woman is retiring tomorrow and she is healthy today. What should she do?

(b) Now assume she has n=2 days till retirement and she is healthy today. What should she do?

(c) Answer The above question for n=3.

(d) Now, write a code to find the optimal policy for n=7.

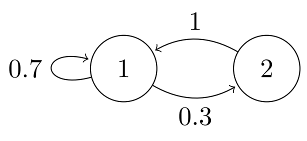
(e) One day, While wandering the wilderness, she comes across a potion. She drinks it and becomes immortal! Now she faces a predicament. She is debating (for the maximum average reward) is it better to perform high-risk stunts every time or perform low-risk stunts all the time (all the times when she is healthy!). Help her decide!

Solution:

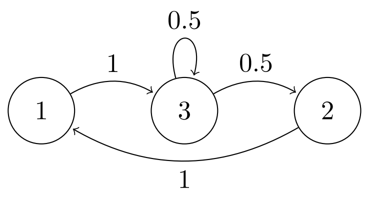
Let $i = 1, 2, 3$ be the state which means the stuntman is "Healthy", has "Minor injuries" and has "Major injuries" respectively. Let L denote the stuntman doing a low-risk stunt and H denote the stuntman doing a high-risk stunt. Let A be the finite set of all possible actions. $A = L, H$. Let a be the possible action in A.

According to the description of the problem, we can form two Markov Chain respectively.

- Action L: do low-risk stunt



- Action H: do high-risk stunt



(a) She should definitely choose to do a high-risk stunt today.

No matter whatever state she will be in tomorrow, she will be retiring and the reward will always be 0 tomorrow. So the strategy is to make rewards as much as possible today, which means she should do a high-risk stunt today.

Mathematically, let's maximize reward in one step when the system is in state 1, its expected reward is

$$V_1^*(1) = Max_a(R(1,a)) = Max(R(1,L), R(1,H)) = Max(100, 400) = 400$$

(b) Let's maximize reward in 2 steps when the system is in state 1, its expected reward is

$$\begin{aligned}V_1^*(2) &= Max_a(\ R(1,a) + P_{11}(a)V_1^*(1) + P_{12}(a)V_2^*(1) + P_{13}(a)V_3^*(1)\ ) \\ &= Max(\ R(1,L) + P_{11}(L)V_1^*(1) + P_{12}(L)V_2^*(1)\ ,\ R(1,H) + P_{13}(H)V_3^*(1)\ ) \\ &= Max(\ 100 + 0.7 \times 400 + 0.3 \times 0\ ,\ 400 + 1 \times 0\ ) \\ &= Max(\ 380, 400\ ) \\ &= 400\end{aligned}$$

Hence, she should do a high-risk stunt today.

(c) Form (b) we know

$$V_1^*(2) = 400$$

And we have

$$V_2^*(2) = V_2^*(1) + V_1^*(1) = 400$$

$$V_3^*(2) = P_{33}V_3^*(1) + P_{33}V_2^*(1) = 0$$

When $n = 3$, let's maximize reward in 3 steps when the system is in state 1, its expected reward is

$$\begin{aligned}V_1^*(3) &= Max_a(\ R(1,a) + P_{11}(a)V_1^*(2) + P_{12}(a)V_2^*(2) + P_{13}(a)V_3^*(2)\ ) \\ &= Max(\ R(1,L) + P_{11}(L)V_1^*(2) + P_{12}(L)V_2^*(2)\ ,\ R(1,H) + P_{13}(H)V_3^*(2)\ ) \\ &= Max(\ 100 + 0.7 \times 400 + 0.3 \times 400\ ,\ 400 + 1 \times 0\ ) \\ &= Max(\ 500, 400\ ) \\ &= 500\end{aligned}$$

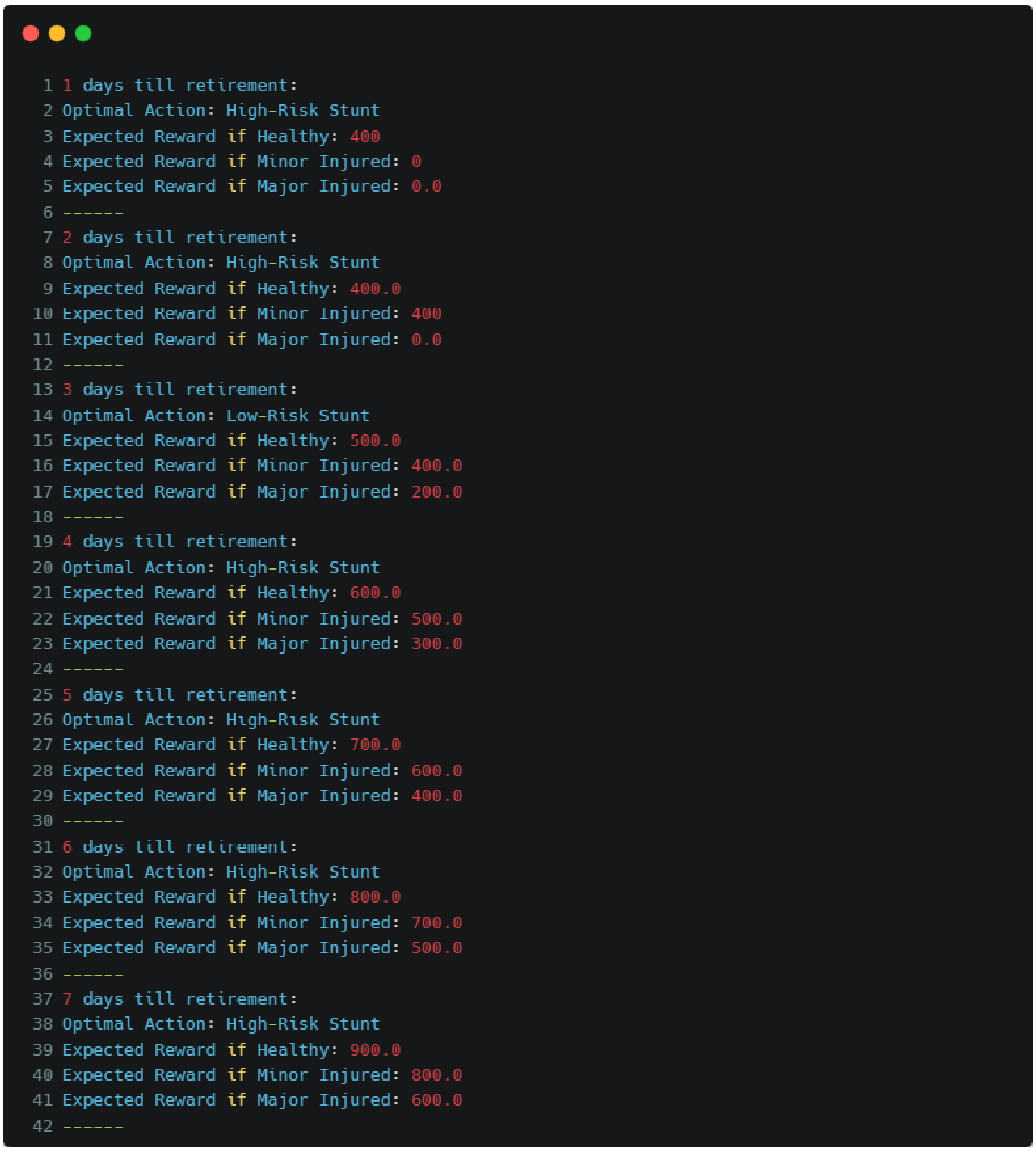
Hence, she should do a low-risk stunt today, and she should do a high-risk stunt if she is healthy tomorrow (with probability 0.7). If she has "Minor injuries" tomorrow (with probability 0.3), she should do a high-risk stunt when she becomes healthy the day after tomorrow.

(d) The optimal policy is: If she is healthy at the day that she has 3 days till retirement, she should do a low-risk stunt, otherwise, keep doing a high-risk stunt when she is healthy.

Then the expected reward $V_1^*(7) = 900$ is optimal

(note: she can do nothing when she has "Minor injuries" or "Major injuries".

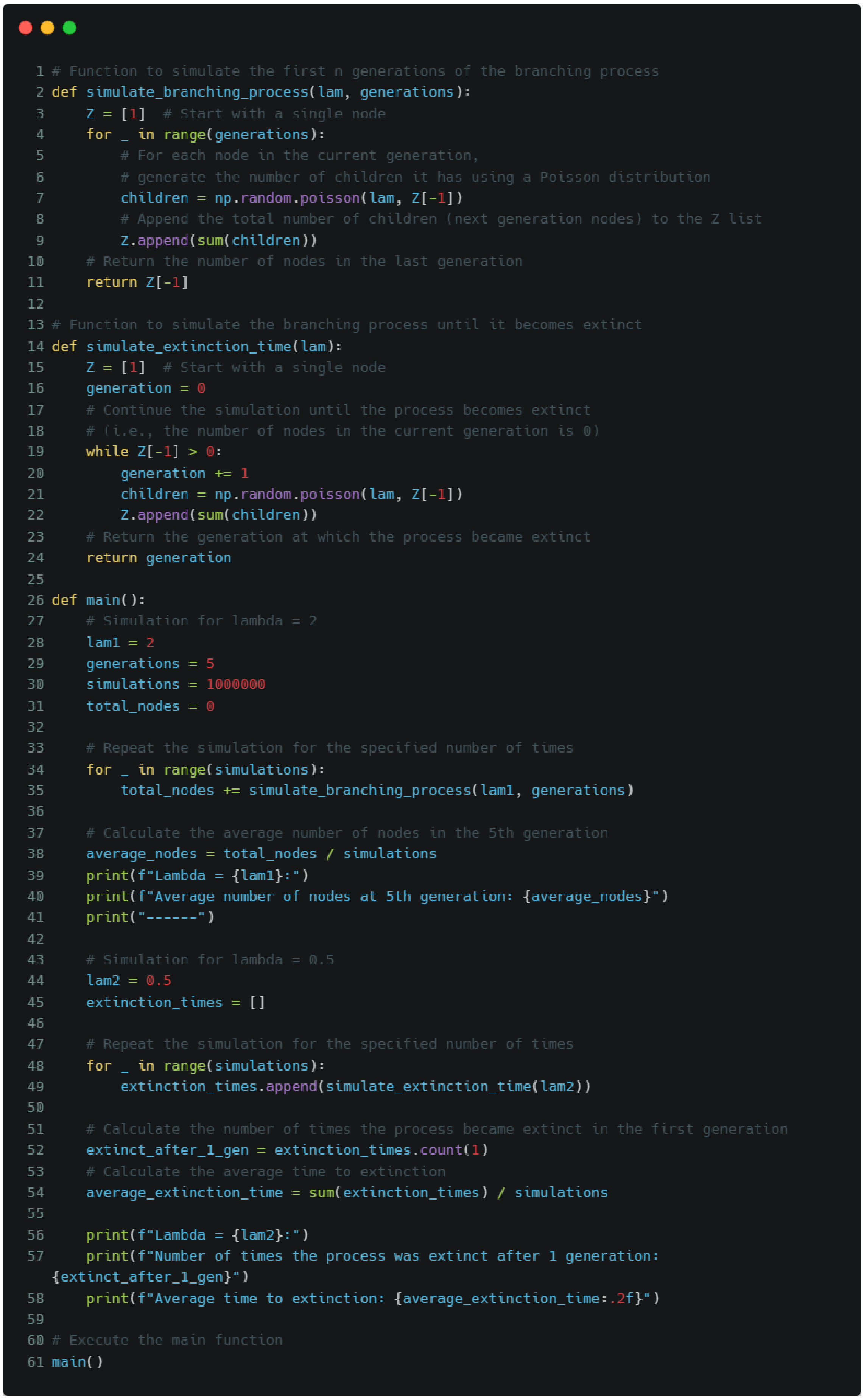
The calculation for n from 1 to 7 is shown below.



```
1 days till retirement:
Optimal Action: High-Risk Stunt
Expected Reward if Healthy: 400
Expected Reward if Minor Injured: 0
Expected Reward if Major Injured: 0.0
------
2 days till retirement:
Optimal Action: High-Risk Stunt
Expected Reward if Healthy: 400.0
Expected Reward if Minor Injured: 400
Expected Reward if Major Injured: 0.0
------
3 days till retirement:
Optimal Action: Low-Risk Stunt
Expected Reward if Healthy: 500.0
Expected Reward if Minor Injured: 400.0
Expected Reward if Major Injured: 200.0
------
4 days till retirement:
Optimal Action: High-Risk Stunt
Expected Reward if Healthy: 600.0
Expected Reward if Minor Injured: 500.0
Expected Reward if Major Injured: 300.0
------
5 days till retirement:
Optimal Action: High-Risk Stunt
Expected Reward if Healthy: 700.0
Expected Reward if Minor Injured: 600.0
Expected Reward if Major Injured: 400.0
------
6 days till retirement:
Optimal Action: High-Risk Stunt
Expected Reward if Healthy: 800.0
Expected Reward if Minor Injured: 700.0
Expected Reward if Major Injured: 500.0
------
7 days till retirement:
Optimal Action: High-Risk Stunt
Expected Reward if Healthy: 900.0
Expected Reward if Minor Injured: 800.0
Expected Reward if Major Injured: 600.0
------
```

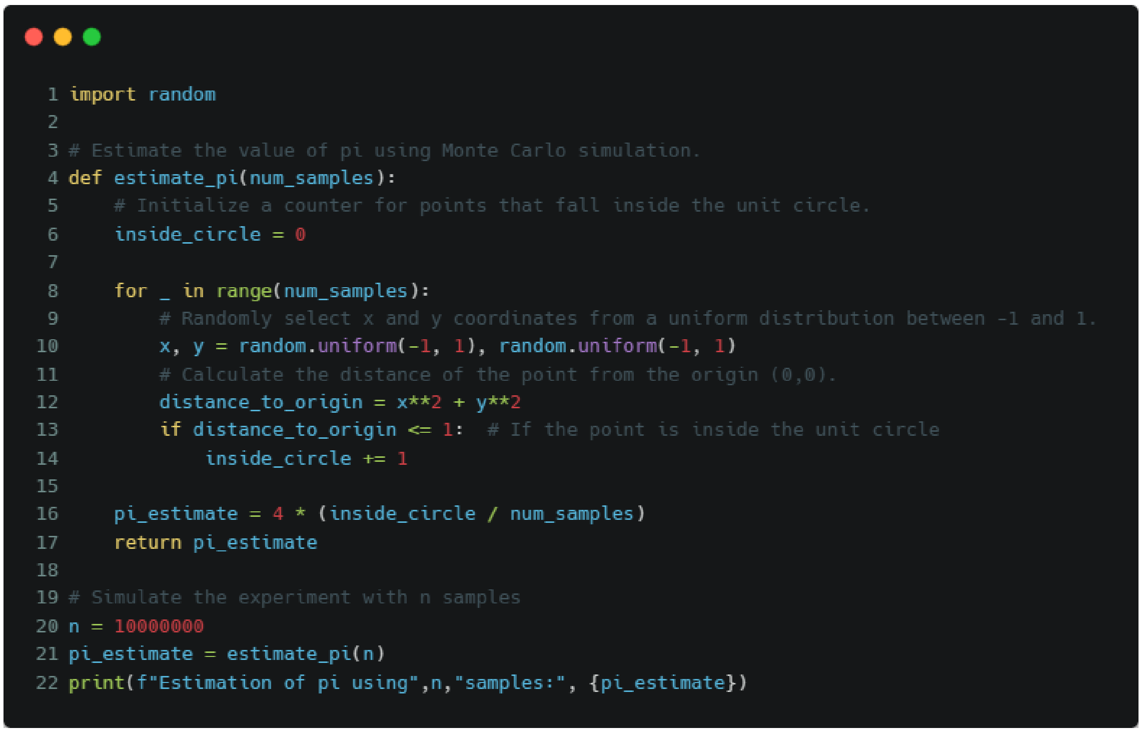
(e) In this case, it means $n = \infty$. It is better for her to perform high-risk stunts every time.

6 Python code for Problem 1



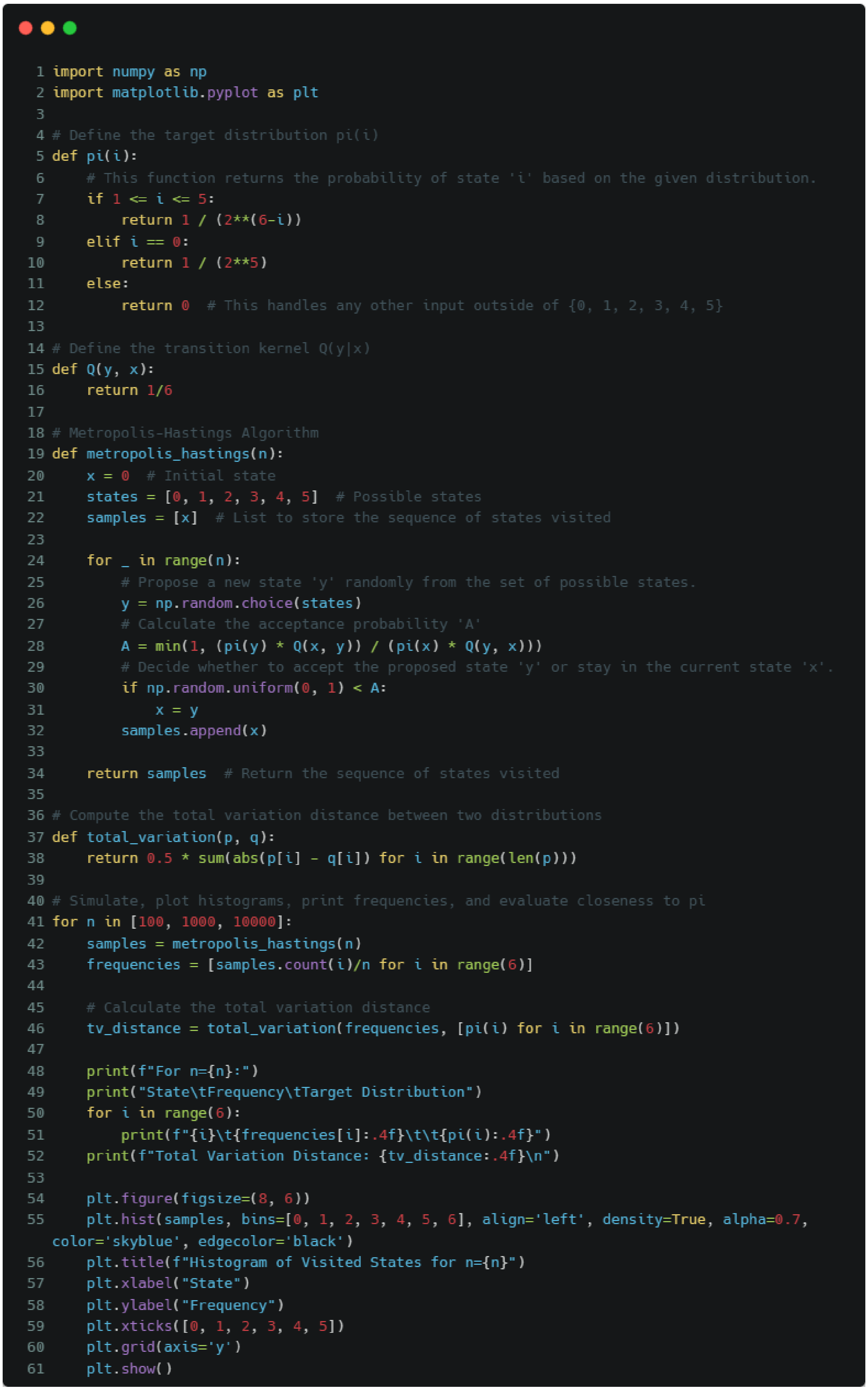
```python
# Function to simulate the first n generations of the branching process
def simulate_branching_process(lam, generations):
    Z = [1]  # Start with a single node
    for _ in range(generations):
        # For each node in the current generation,
        # generate the number of children it has using a Poisson distribution
        children = np.random.poisson(lam, Z[-1])
        # Append the total number of children (next generation nodes) to the Z list
        Z.append(sum(children))
    # Return the number of nodes in the last generation
    return Z[-1]

# Function to simulate the branching process until it becomes extinct
def simulate_extinction_time(lam):
    Z = [1]  # Start with a single node
    generation = 0
    # Continue the simulation until the process becomes extinct
    # (i.e., the number of nodes in the current generation is 0)
    while Z[-1] > 0:
        generation += 1
        children = np.random.poisson(lam, Z[-1])
        Z.append(sum(children))
    # Return the generation at which the process became extinct
    return generation

def main():
    # Simulation for lambda = 2
    lam1 = 2
    generations = 5
    simulations = 1000000
    total_nodes = 0

    # Repeat the simulation for the specified number of times
    for _ in range(simulations):
        total_nodes += simulate_branching_process(lam1, generations)

    # Calculate the average number of nodes in the 5th generation
    average_nodes = total_nodes / simulations
    print(f"Lambda = {lam1}:")
    print(f"Average number of nodes at 5th generation: {average_nodes}")
    print("------")

    # Simulation for lambda = 0.5
    lam2 = 0.5
    extinction_times = []

    # Repeat the simulation for the specified number of times
    for _ in range(simulations):
        extinction_times.append(simulate_extinction_time(lam2))

    # Calculate the number of times the process became extinct in the first generation
    extinct_after_1_gen = extinction_times.count(1)
    # Calculate the average time to extinction
    average_extinction_time = sum(extinction_times) / simulations

    print(f"Lambda = {lam2}:")
    print(f"Number of times the process was extinct after 1 generation: {extinct_after_1_gen}")
    print(f"Average time to extinction: {average_extinction_time:.2f}")

# Execute the main function
main()
```

7 Python code for Problem 3



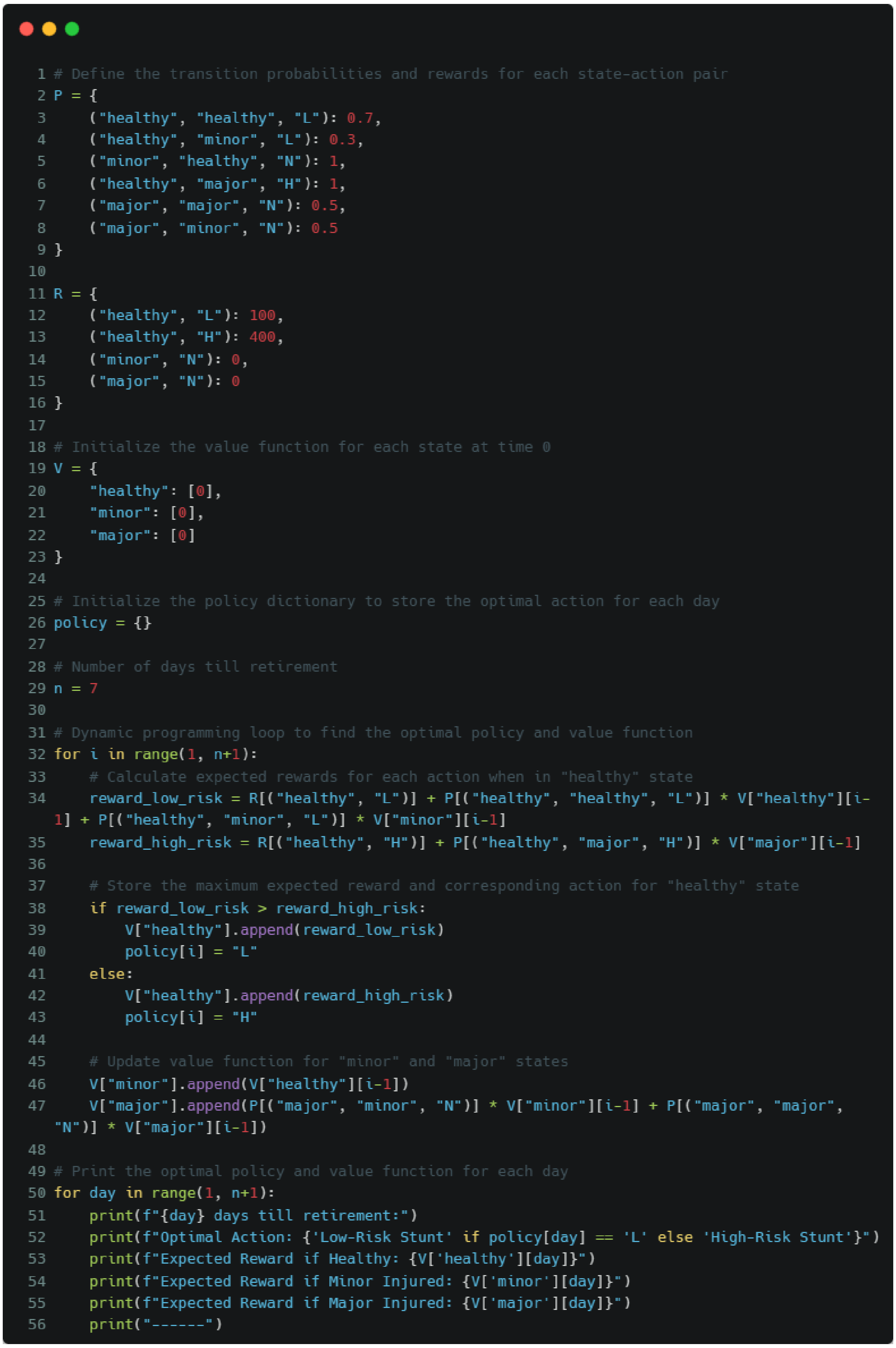
```python
import random

# Estimate the value of pi using Monte Carlo simulation.
def estimate_pi(num_samples):
    # Initialize a counter for points that fall inside the unit circle.
    inside_circle = 0

    for _ in range(num_samples):
        # Randomly select x and y coordinates from a uniform distribution between -1 and 1.
        x, y = random.uniform(-1, 1), random.uniform(-1, 1)
        # Calculate the distance of the point from the origin (0,0).
        distance_to_origin = x**2 + y**2
        if distance_to_origin <= 1:  # If the point is inside the unit circle
            inside_circle += 1

    pi_estimate = 4 * (inside_circle / num_samples)
    return pi_estimate

# Simulate the experiment with n samples
n = 10000000
pi_estimate = estimate_pi(n)
print(f"Estimation of pi using",n,"samples:", {pi_estimate})
```

8 Python code for Problem 4



```python
import numpy as np
import matplotlib.pyplot as plt

# Define the target distribution pi(i)
def pi(i):
    # This function returns the probability of state 'i' based on the given distribution.
    if 1 <= i <= 5:
        return 1 / (2**(6-i))
    elif i == 0:
        return 1 / (2**5)
    else:
        return 0  # This handles any other input outside of {0, 1, 2, 3, 4, 5}

# Define the transition kernel Q(y|x)
def Q(y, x):
    return 1/6

# Metropolis-Hastings Algorithm
def metropolis_hastings(n):
    x = 0  # Initial state
    states = [0, 1, 2, 3, 4, 5]  # Possible states
    samples = [x]  # List to store the sequence of states visited

    for _ in range(n):
        # Propose a new state 'y' randomly from the set of possible states.
        y = np.random.choice(states)
        # Calculate the acceptance probability 'A'
        A = min(1, (pi(y) * Q(x, y)) / (pi(x) * Q(y, x)))
        # Decide whether to accept the proposed state 'y' or stay in the current state 'x'.
        if np.random.uniform(0, 1) < A:
            x = y
        samples.append(x)

    return samples  # Return the sequence of states visited

# Compute the total variation distance between two distributions
def total_variation(p, q):
    return 0.5 * sum(abs(p[i] - q[i]) for i in range(len(p)))

# Simulate, plot histograms, print frequencies, and evaluate closeness to pi
for n in [100, 1000, 10000]:
    samples = metropolis_hastings(n)
    frequencies = [samples.count(i)/n for i in range(6)]

    # Calculate the total variation distance
    tv_distance = total_variation(frequencies, [pi(i) for i in range(6)])

    print(f"For n={n}:")
    print("State\tFrequency\tTarget Distribution")
    for i in range(6):
        print(f"{i}\t{frequencies[i]:.4f}\t\t{pi(i):.4f}")
    print(f"Total Variation Distance: {tv_distance:.4f}\n")

    plt.figure(figsize=(8, 6))
    plt.hist(samples, bins=[0, 1, 2, 3, 4, 5, 6], align='left', density=True, alpha=0.7, color='skyblue', edgecolor='black')
    plt.title(f"Histogram of Visited States for n={n}")
    plt.xlabel("State")
    plt.ylabel("Frequency")
    plt.xticks([0, 1, 2, 3, 4, 5])
    plt.grid(axis='y')
    plt.show()
```

9 Python code for Problem 5



```python
# Define the transition probabilities and rewards for each state-action pair
P = {
    ("healthy", "healthy", "L"): 0.7,
    ("healthy", "minor", "L"): 0.3,
    ("minor", "healthy", "N"): 1,
    ("healthy", "major", "H"): 1,
    ("major", "major", "N"): 0.5,
    ("major", "minor", "N"): 0.5
}

R = {
    ("healthy", "L"): 100,
    ("healthy", "H"): 400,
    ("minor", "N"): 0,
    ("major", "N"): 0
}

# Initialize the value function for each state at time 0
V = {
    "healthy": [0],
    "minor": [0],
    "major": [0]
}

# Initialize the policy dictionary to store the optimal action for each day
policy = {}

# Number of days till retirement
n = 7

# Dynamic programming loop to find the optimal policy and value function
for i in range(1, n+1):
    # Calculate expected rewards for each action when in "healthy" state
    reward_low_risk = R[("healthy", "L")] + P[("healthy", "healthy", "L")] * V["healthy"][i-1] + P[("healthy", "minor", "L")] * V["minor"][i-1]
    reward_high_risk = R[("healthy", "H")] + P[("healthy", "major", "H")] * V["major"][i-1]

    # Store the maximum expected reward and corresponding action for "healthy" state
    if reward_low_risk > reward_high_risk:
        V["healthy"].append(reward_low_risk)
        policy[i] = "L"
    else:
        V["healthy"].append(reward_high_risk)
        policy[i] = "H"

    # Update value function for "minor" and "major" states
    V["minor"].append(V["healthy"][i-1])
    V["major"].append(P[("major", "minor", "N")] * V["minor"][i-1] + P[("major", "major", "N")] * V["major"][i-1])

# Print the optimal policy and value function for each day
for day in range(1, n+1):
    print(f"{day} days till retirement:")
    print(f"Optimal Action: {'Low-Risk Stunt' if policy[day] == 'L' else 'High-Risk Stunt'}")
    print(f"Expected Reward if Healthy: {V['healthy'][day]}")
    print(f"Expected Reward if Minor Injured: {V['minor'][day]}")
    print(f"Expected Reward if Major Injured: {V['major'][day]}")
    print("------")
```