Assignment 2 Report

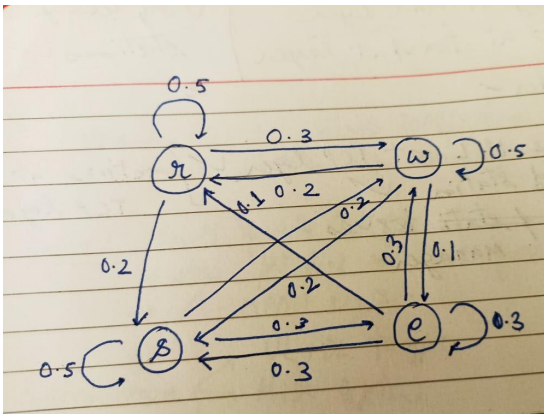
Srimant Mohanty
2021207
srimant21207@iiitd.ac.in

Q1. **Markov Chain**

Writing of a paper can be studied as a Finite State Discrete Time Markov Chain since it has 4 finite states: read(r) , write(w) , e-mail(e) , surf(s) & the markov chain evolves in discrete units of time(in minutes). Further, the initial one step probability matrix is given as the following:-

$$P = \begin{array}{c} \\ r \\ w \\ e \\ s \end{array} \begin{array}{c} \begin{array}{cccc} r & w & e & s \end{array} \\ \begin{pmatrix} 0.5 & 0.3 & 0 & 0.2 \\ 0.2 & 0.5 & 0.1 & 0.2 \\ 0.1 & 0.3 & 0.3 & 0.3 \\ 0 & 0.2 & 0.3 & 0.5 \end{pmatrix} \end{array}$$

The Transition Diagram for this Markov Chain can be constructed as follows, this will help us visualize the problem better



a) To study how the Markov chain evolves over 20 minutes, we can simply multiply the matrix 20 times with itself since the transition matrix raised to a power represents the state transitions after that many time steps and the ‘i’ th row and the ‘j’ th column of the matrix represents the probability of transitioning from i to j state in the markov chain.

Further, the evolution of the markov chain will also depend on the initial state of the markov chain. In order to understand how the markov chain evolves at every step we simulate it 4 times with the initial states taken as ‘read’ , ‘write’ , ‘email’ & ‘surf’ respectively.

The Markov Chain Transition Probability matrix after 20 minutes:-

```
[[0.17073182 0.33604338 0.18157173 0.31165307],
 [0.17073177 0.33604337 0.18157177 0.31165309],
 [0.17073168 0.33604336 0.18157183 0.31165313],
 [0.17073159 0.33604334 0.1815719 0.31165317]]
```

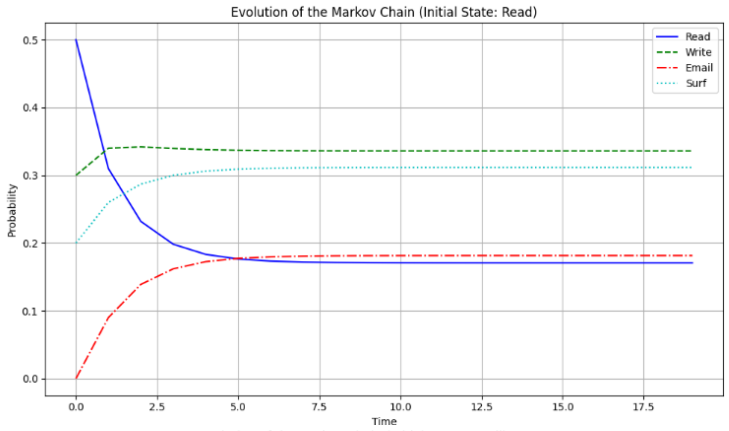
	r	w	e	s
r	0.17073182	0.33604338	0.18157173	0.31165307
w	0.17073177	0.33604337	0.18157177	0.31165309
e	0.17073168	0.33604336	0.18157183	0.31165313
s	0.17073159	0.33604334	0.1815719	0.31165317

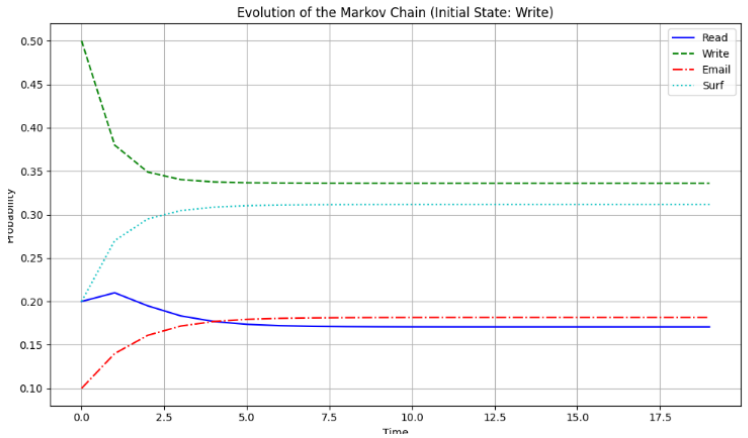
Code:

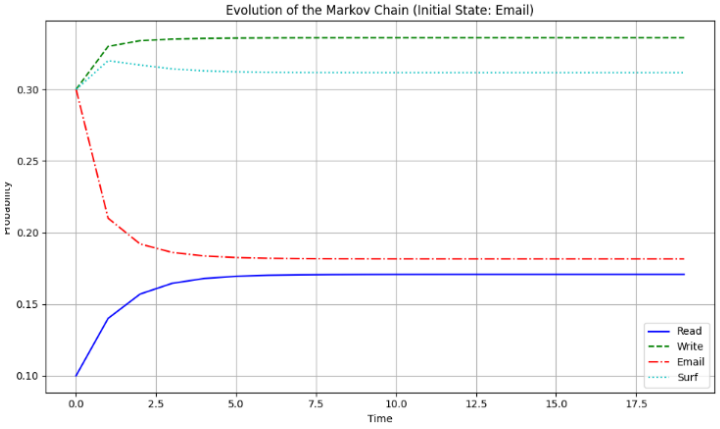
```python
import numpy as np
import matplotlib.pyplot as plt

# Transition matrix
P = np.array([[0.5, 0.3, 0, 0.2],
              [0.2, 0.5, 0.1, 0.2],
              [0.1, 0.3, 0.3, 0.3],
              [0, 0.2, 0.3, 0.5]])

# Define states and corresponding labels
states = ["Read", "Write", "Email", "Surf"]

# Define line styles and colors for plotting
line = ['-', '--', '-.', ':']
colors = ['b', 'g', 'r', 'c']

# Loop through each initial state
for j in range(4):
    initial_state = np.zeros(4)
    initial_state[j] = 1

    # Evolve the state
    evolution = []
    for i in range(20):
        initial_state = np.dot(initial_state, P)
        print("Time:",i+1)
        evolution.append(initial_state)
        print(evolution[-1])

    # Plot the evolution of each state
    plt.figure(figsize=(10, 6))
    for idx, state_idx in enumerate(range(4)):
        plt.plot([evolution[i][state_idx] for i in range(20)], label=f"{states[state_idx]}", linestyle=line[idx], color=colors[idx])

    plt.xlabel('Time')
    plt.ylabel('Probability')
    plt.title(f'Evolution of the Markov Chain (Initial State: {states[j]})')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```

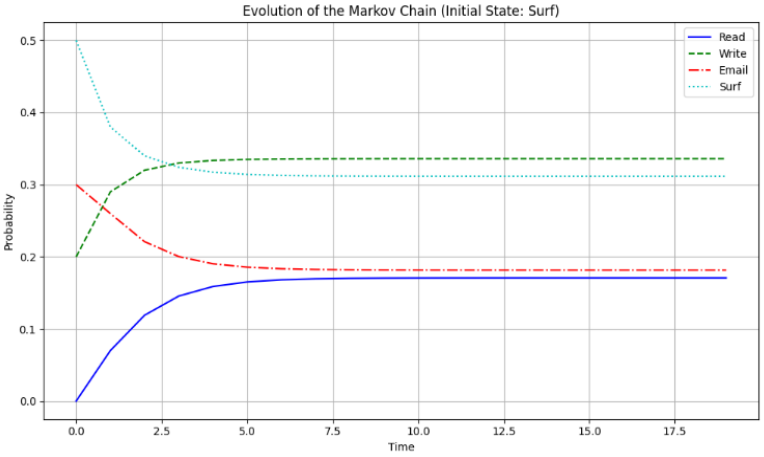
The following plots show how the markov chain evolves in 20 minutes depending on the initial state chosen i.e. read, write, email or surf.

Plots:









Initial State: Read

```
Time: 1
[0.5 0.3 0.  0.2]
Time: 2
[0.31 0.34 0.09 0.26]
Time: 3
[0.232 0.342 0.139 0.287]
Time: 4
[0.1983 0.3397 0.162  0.3   ]
Time: 5
[0.18329 0.33794 0.17257 0.3062 ]
Time: 6
[0.17649  0.336968 0.177425 0.309117]
Time: 7
[0.1733811 0.3364819 0.1796594 0.3104776]
Time: 8
[0.17195287 0.33624862 0.18068929 0.31110922]
Time: 9
[0.17129509 0.3361388  0.18116441 0.3114017 ]
Time: 10
[0.17099175 0.33608759 0.18138371 0.31153695]
Time: 11
[0.17085176 0.33606382 0.18148496 0.31159946]
Time: 12
[0.17078714 0.33605282 0.18153171 0.31162833]
Time: 13
[0.17075731 0.33604773 0.18155329 0.31164167]
Time: 14
[0.17074353 0.33604538 0.18156326 0.31164783]
Time: 15
[0.17073717 0.33604429 0.18156787 0.31165068]
Time: 16
[0.17073423 0.33604379 0.18156999 0.31165199]
Time: 17
[0.17073287 0.33604356 0.18157097 0.3116526 ]
Time: 18
[0.17073224 0.33604345 0.18157143 0.31165288]
Time: 19
[0.17073196 0.3360434  0.18157164 0.31165301]
Time: 20
[0.17073182 0.33604338 0.18157173 0.31165307]
```

Initial State: Write

```
Time: 1
[0.2 0.5 0.1 0.2]
Time: 2
[0.21 0.38 0.14 0.27]
Time: 3
[0.195 0.349 0.161 0.295]
Time: 4
[0.1834 0.3403 0.1717 0.3046]
Time: 5
[0.17693 0.3376  0.17692 0.30855]
Time: 6
[0.173677 0.336665 0.179401 0.310257]
Time: 7
[0.1721116 0.3363073 0.1805639 0.3110172]
Time: 8
[0.17137365 0.33615974 0.18110506 0.31136155]
Time: 9
[0.17102928 0.33609579 0.18135596 0.31151897]
Time: 10
[0.17086939 0.33606726 0.18147206 0.31159129]
Time: 11
[0.17079535 0.33605432 0.18152573 0.31162459]
Time: 12
[0.17076112 0.33604841 0.18155053 0.31163995]
Time: 13
[0.17074529 0.33604569 0.18156198 0.31164704]
Time: 14
[0.17073798 0.33604443 0.18156728 0.31165031]
Time: 15
[0.1707346  0.33604386 0.18156972 0.31165182]
Time: 16
[0.17073305 0.33604359 0.18157085 0.31165252]
Time: 17
[0.17073233 0.33604347 0.18157137 0.31165284]
Time: 18
[0.17073199 0.33604341 0.18157161 0.31165299]
Time: 19
[0.17073184 0.33604338 0.18157172 0.31165306]
Time: 20
[0.17073177 0.33604337 0.18157177 0.31165309]
```

Initial State: E-mail

```
Time: 1
[0.  0.2 0.3 0.5]
Time: 2
[0.07 0.29 0.26 0.38]
Time: 3
[0.119 0.32  0.221 0.34 ]
Time: 4
[0.1456 0.33   0.2003 0.3241]
Time: 5
[0.15883 0.33359 0.19032 0.31726]
Time: 6
[0.165165 0.334992 0.185633 0.31421 ]
Time: 7
[0.1681442 0.3355774 0.1834521 0.3128263]
Time: 8
[0.16953279 0.33583285 0.18244126 0.3121931 ]
Time: 9
[0.17017709 0.33594726 0.18197359 0.31190206]
Time: 10
[0.17047536 0.33599925 0.18175742 0.31176798]
Time: 11
[0.17061327 0.33602305 0.18165754 0.31170613]
Time: 12
[0.170677   0.336034   0.18161141 0.31167759]
Time: 13
[0.17070644 0.33603904 0.1815901  0.31166442]
Time: 14
[0.17072004 0.33604137 0.18158026 0.31165834]
Time: 15
[0.17072632 0.33604244 0.18157572 0.31165553]
Time: 16
[0.17072922 0.33604294 0.18157362 0.31165423]
Time: 17
[0.17073056 0.33604316 0.18157265 0.31165363]
Time: 18
[0.17073118 0.33604327 0.1815722  0.31165335]
Time: 19
[0.17073146 0.33604332 0.18157199 0.31165323]
Time: 20
[0.17073159 0.33604334 0.1815719  0.31165317]
```

Initial State: Surf

```
Time: 1
[0.1 0.3 0.3 0.3]
Time: 2
[0.14 0.33 0.21 0.32]
Time: 3
[0.157 0.334 0.192 0.317]
Time: 4
[0.1645 0.3351 0.1861 0.3143]
Time: 5
[0.16788 0.33559 0.18363 0.3129 ]
Time: 6
[0.169421 0.335828 0.182518 0.312233]
Time: 7
[0.1701279 0.3359423 0.1820081 0.3119217]
Time: 8
[0.17045322 0.33599629 0.18177317 0.31177732]
Time: 9
[0.17060318 0.33602153 0.18166478 0.31171051]
Time: 10
[0.17067238 0.33603325 0.18161474 0.31167963]
Time: 11
[0.17070431 0.33603869 0.18159164 0.31166536]
Time: 12
[0.17071906 0.3360412  0.18158097 0.31165877]
Time: 13
[0.17072587 0.33604236 0.18157604 0.31165573]
Time: 14
[0.17072901 0.3360429  0.18157377 0.31165432]
Time: 15
[0.17073046 0.33604315 0.18157272 0.31165367]
Time: 16
[0.17073113 0.33604326 0.18157223 0.31165337]
Time: 17
[0.17073144 0.33604332 0.18157201 0.31165324]
Time: 18
[0.17073158 0.33604334 0.1815719  0.31165317]
Time: 19
[0.17073165 0.33604335 0.18157186 0.31165314]
Time: 20
[0.17073168 0.33604336 0.18157183 0.31165313]
```

P(X20 =s | X0 = r) is equal to the ‘r’ th row and ‘s’ th column in the transition matrix at the 20th step/20 minutes. This can be understood from the equation:-

$$P(X20 = s \mid X0 = r) = \pi_0 * P^{20}$$

Code:

```
#probability of P20 = 'surf' given that P0 = 'read'
P20 = np.linalg.matrix_power(P,20)
print('Markov Chain after 20 minutes:')
print(P20)
print("\n")
print('Probability of reaching surf state after 20 minutes:')
print(P20[0,3])
```

Result:

```
Markov Chain after 20 minutes:
[[0.17073182 0.33604338 0.18157173 0.31165307]
 [0.17073177 0.33604337 0.18157177 0.31165309]
 [0.17073168 0.33604336 0.18157183 0.31165313]
 [0.17073159 0.33604334 0.1815719  0.31165317]]


Probability of reaching surf state after 20 minutes:
0.3116530652582661
```

Therefore, the probability of reaching ‘surf’ from ‘read’ after 20 minutes is 0.3116530652582661

b) Similar to a) part we now show how the markov chain evolves after 25 minutes.

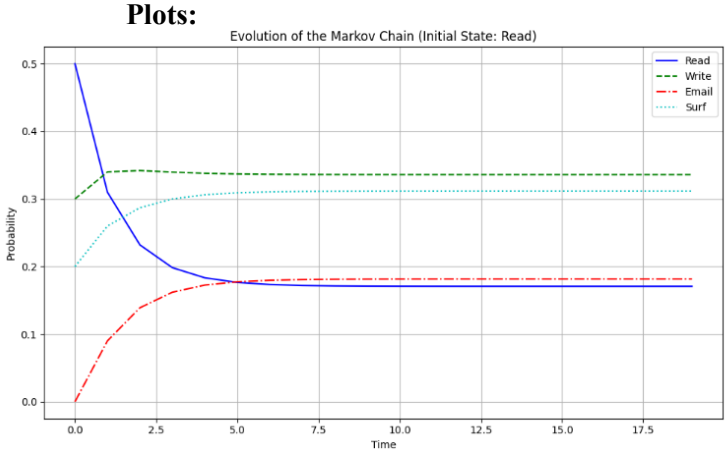
The Markov Chain Transition Probability matrix after 20 minutes:-

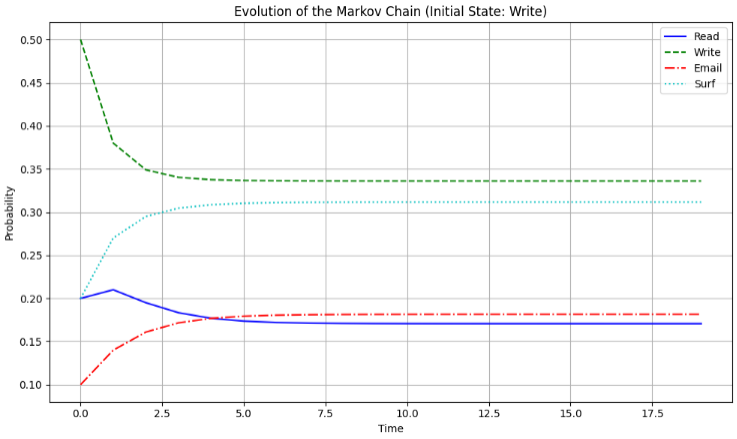
```
[[0.17073171 0.33604336 0.18157181 0.31165312],
 [0.17073171 0.33604336 0.18157181 0.31165312],
 [0.17073171 0.33604336 0.18157182 0.31165312],
 [0.1707317 0.33604336 0.18157182 0.31165312]]
```

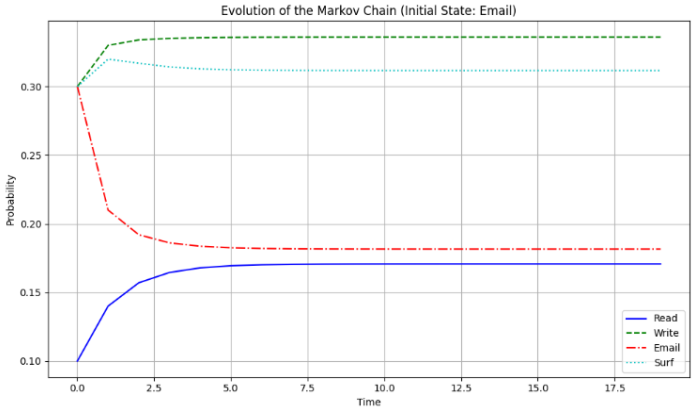
	r	w	e	s
r	0.17073171	0.33604336	0.18157181	0.31165312
w	0.17073171	0.33604336	0.18157181	0.31165312
e	0.17073171	0.33604336	0.18157182	0.31165312
s	0.1707317	0.33604336	0.18157182	0.31165312

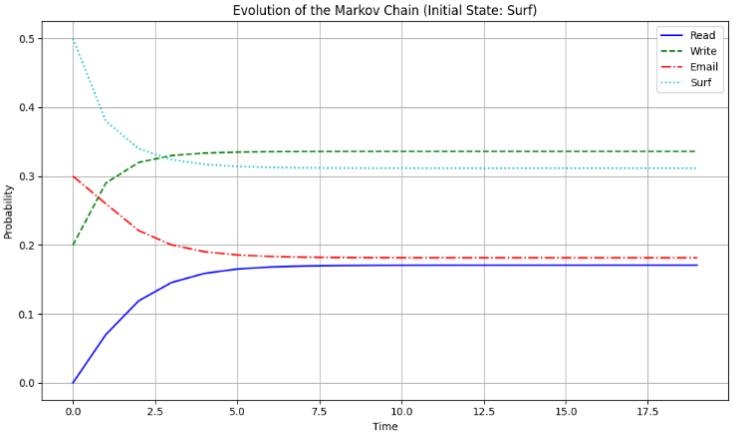
The following plots show how the markov chain evolves in 25 minutes depending on the initial state chosen i.e. read, write, email or surf.

Plots:









Probabilities after every minute:

Initial State: Read

```
Time: 1
[0.5 0.3 0.  0.2]
Time: 2
[0.31 0.34 0.09 0.26]
Time: 3
[0.232 0.342 0.139 0.287]
Time: 4
[0.1983 0.3397 0.162  0.3   ]
Time: 5
[0.18329 0.33794 0.17257 0.3062 ]
Time: 6
[0.17649  0.336968 0.177425 0.309117]
Time: 7
[0.1733811 0.3364819 0.1796594 0.3104776]
Time: 8
[0.17195287 0.33624862 0.18068929 0.31110922]
Time: 9
[0.17129509 0.3361388  0.18116441 0.3114017 ]
Time: 10
[0.17099175 0.33608759 0.18138371 0.31153695]
Time: 11
[0.17085176 0.33606382 0.18148496 0.31159946]
Time: 12
[0.17078714 0.33605282 0.18153171 0.31162833]
Time: 13
[0.17075731 0.33604773 0.18155329 0.31164167]
Time: 14
[0.17074353 0.33604538 0.18156326 0.31164783]
Time: 15
[0.17073717 0.33604429 0.18156787 0.31165068]
Time: 16
[0.17073423 0.33604379 0.18156999 0.31165199]
Time: 17
[0.17073287 0.33604356 0.18157097 0.3116526 ]
Time: 18
[0.17073224 0.33604345 0.18157143 0.31165288]
Time: 19
[0.17073196 0.3360434  0.18157164 0.31165301]
Time: 20
[0.17073182 0.33604338 0.18157173 0.31165307]
Time: 21
[0.17073176 0.33604337 0.18157178 0.31165309]
Time: 22
[0.17073173 0.33604336 0.1815718  0.31165311]
Time: 23
[0.17073172 0.33604336 0.18157181 0.31165311]
Time: 24
[0.17073171 0.33604336 0.18157181 0.31165311]
Time: 25
[0.17073171 0.33604336 0.18157181 0.31165312]
```

Initial State: Write

```
Time: 1
[0.2 0.5 0.1 0.2]
Time: 2
[0.21 0.38 0.14 0.27]
Time: 3
[0.195 0.349 0.161 0.295]
Time: 4
[0.1834 0.3403 0.1717 0.3046]
Time: 5
[0.17693 0.3376  0.17692 0.30855]
Time: 6
[0.173677 0.336665 0.179401 0.310257]
Time: 7
[0.1721116 0.3363073 0.1805639 0.3110172]
Time: 8
[0.17137365 0.33615974 0.18110506 0.31136155]
Time: 9
[0.17102928 0.33609579 0.18135596 0.31151897]
Time: 10
[0.17086939 0.33606726 0.18147206 0.31159129]
Time: 11
[0.17079535 0.33605432 0.18152573 0.31162459]
Time: 12
[0.17076112 0.33604841 0.18155053 0.31163995]
Time: 13
[0.17074529 0.33604569 0.18156198 0.31164704]
Time: 14
[0.17073798 0.33604443 0.18156728 0.31165031]
Time: 15
[0.1707346  0.33604386 0.18156972 0.31165182]
Time: 16
[0.17073305 0.33604359 0.18157085 0.31165252]
Time: 17
[0.17073233 0.33604347 0.18157137 0.31165284]
Time: 18
[0.17073199 0.33604341 0.18157161 0.31165299]
Time: 19
[0.17073184 0.33604338 0.18157172 0.31165306]
Time: 20
[0.17073177 0.33604337 0.18157177 0.31165309]
Time: 21
[0.17073174 0.33604337 0.1815718  0.3116531 ]
Time: 22
[0.17073172 0.33604336 0.18157181 0.31165311]
Time: 23
[0.17073171 0.33604336 0.18157181 0.31165311]
Time: 24
[0.17073171 0.33604336 0.18157181 0.31165312]
Time: 25
[0.17073171 0.33604336 0.18157181 0.31165312]
```

Initial State: E-mail

```
Time: 1
[0.1 0.3 0.3 0.3]
Time: 2
[0.14 0.33 0.21 0.32]
Time: 3
[0.157 0.334 0.192 0.317]
Time: 4
[0.1645 0.3351 0.1861 0.3143]
Time: 5
[0.16788 0.33559 0.18363 0.3129 ]
Time: 6
[0.169421 0.335828 0.182518 0.312233]
Time: 7
[0.1701279 0.3359423 0.1820081 0.3119217]
Time: 8
[0.17045322 0.33599629 0.18177317 0.31177732]
Time: 9
[0.17060318 0.33602153 0.18166478 0.31171051]
Time: 10
[0.17067238 0.33603325 0.18161474 0.31167963]
Time: 11
[0.17070431 0.33603869 0.18159164 0.31166536]
Time: 12
[0.17071906 0.3360412  0.18158097 0.31165877]
Time: 13
[0.17072587 0.33604236 0.18157604 0.31165573]
Time: 14
[0.17072901 0.3360429  0.18157377 0.31165432]
Time: 15
[0.17073046 0.33604315 0.18157272 0.31165367]
Time: 16
[0.17073113 0.33604326 0.18157223 0.31165337]
Time: 17
[0.17073144 0.33604332 0.18157201 0.31165324]
Time: 18
[0.17073158 0.33604334 0.1815719  0.31165317]
Time: 19
[0.17073165 0.33604335 0.18157186 0.31165314]
Time: 20
[0.17073168 0.33604336 0.18157183 0.31165313]
Time: 21
[0.1707317  0.33604336 0.18157182 0.31165312]
Time: 22
[0.1707317  0.33604336 0.18157182 0.31165312]
Time: 23
[0.1707317  0.33604336 0.18157182 0.31165312]
Time: 24
[0.17073171 0.33604336 0.18157182 0.31165312]
Time: 25
[0.17073171 0.33604336 0.18157182 0.31165312]
```

Initial State: Surf

```
Time: 1
[0.  0.2 0.3 0.5]
Time: 2
[0.07 0.29 0.26 0.38]
Time: 3
[0.119 0.32  0.221 0.34 ]
Time: 4
[0.1456 0.33   0.2003 0.3241]
Time: 5
[0.15883 0.33359 0.19032 0.31726]
Time: 6
[0.165165 0.334992 0.185633 0.31421 ]
Time: 7
[0.1681442 0.3355774 0.1834521 0.3128263]
Time: 8
[0.16953279 0.33583285 0.18244126 0.3121931 ]
Time: 9
[0.17017709 0.33594726 0.18197359 0.31190206]
Time: 10
[0.17047536 0.33599925 0.18175742 0.31176798]
Time: 11
[0.17061327 0.33602305 0.18165754 0.31170613]
Time: 12
[0.170677   0.336034   0.18161141 0.31167759]
Time: 13
[0.17070644 0.33603904 0.1815901  0.31166442]
Time: 14
[0.17072004 0.33604137 0.18158026 0.31165834]
Time: 15
[0.17072632 0.33604244 0.18157572 0.31165553]
Time: 16
[0.17072922 0.33604294 0.18157362 0.31165423]
Time: 17
[0.17073056 0.33604316 0.18157265 0.31165363]
Time: 18
[0.17073118 0.33604327 0.1815722  0.31165335]
Time: 19
[0.17073146 0.33604332 0.18157199 0.31165323]
Time: 20
[0.17073159 0.33604334 0.1815719  0.31165317]
Time: 21
[0.17073166 0.33604335 0.18157185 0.31165314]
Time: 22
[0.17073168 0.33604336 0.18157183 0.31165313]
Time: 23
[0.1707317  0.33604336 0.18157182 0.31165312]
Time: 24
[0.1707317  0.33604336 0.18157182 0.31165312]
Time: 25
[0.1707317  0.33604336 0.18157182 0.31165312]
```

P(X25 =s | X20 = s) is equal to the ‘s’ th row and ‘s’ th column in the transition matrix at the 25th step/25 minutes. This can be understood from the equation:-

$$P(X25 = s \mid X20 = s) = \pi_{20} * P^{5} = \mathbf{0.31726}$$

Code:

```
print('Probability of reaching surf state after 25 minutes given that 20h state is surf:')
#find P(X25 = s|X20 = s).
pi20 = np.array([0,0,0,1])
P5 = np.linalg.matrix_power(P,5)
#multiply pi20 and P5
pi25 = np.dot(pi20,P5)
print(pi25[-1])
```

Result:

```
Probability of reaching surf state after 25 minutes given that 20h state is surf:
0.31726
```

c) Yes, a stationary distribution exists since we can see that the values are saturating after larger values of time.
Further, the stationary distribution can be calculated by solving the equations:-
1) $\pi = \pi P$ 2) $\Sigma\pi = 1$

This can be done by finding the eigenvalues and eigenvectors of the transpose of the transition matrix , and then finding those eigenvectors with eigenvalues closest to 1 and later normalizing the vectors to convert them into probability vectors.

```
P = np.array([[0.5,0.3,0,0.2],[0.2,0.5,0.1,0.2],[0.1,0.3,0.3,0.3],[0,0.2,0.3,0.5]])
#stationary distribution of the markov chain

#transpose P
P_transpose = np.transpose(P)

#find eigen values and eigen vectors of P_transpose
eigen_values, eigen_vectors = np.linalg.eig(P_transpose)

#Find the indexes of the eigenvalues that are close to one
index = []
for i in range(len(eigen_values)):
    if(np.isclose(eigen_values[i],1)):
        index.append(i)

#find the eigen vectors corresponding to the eigen values that are close to one
eigen_vector = []
for i in range(len(index)):
    eigen_vector.append(eigen_vectors[:,index[i]])

eigen_vector = np.array(eigen_vector)
eigen_vector = np.transpose(eigen_vector)

#normalize the eigen vector
eigen_vector = eigen_vector/np.sum(eigen_vector)

#find the stationary distribution of the markov chain
print(eigen_vector)
```

Stationary Distribution:-

```
[[0.17073171] [0.33604336] [0.18157182] [0.31165312]]
```

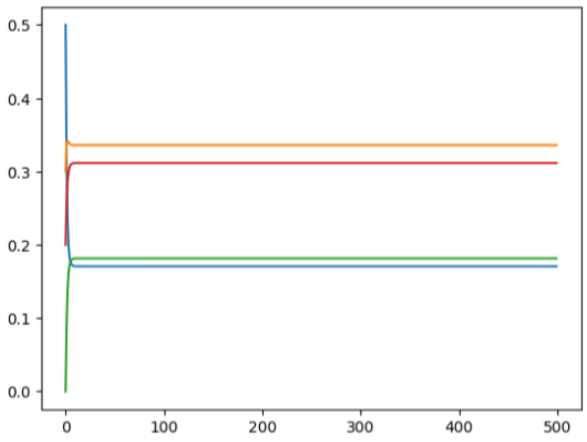
```
The stationary distribution obtained is:
[[0.17073171]
 [0.33604336]
 [0.18157182]
 [0.31165312]]
```

d) Yes, a limiting distribution exists since we can see that the markov chain is aperiodic and irreducible/ This can be seen from the transition probability matrix drawn in the beginning. For such markov chains we know that a unique solution exists and the limiting distribution is the same as the stationary distribution.

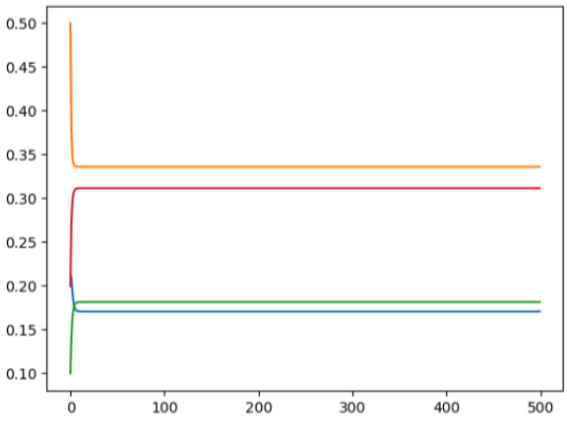
This can also be verified from running the simulations for very large values of time and see that the markov chain converges.

Therefore, the limiting distribution is:

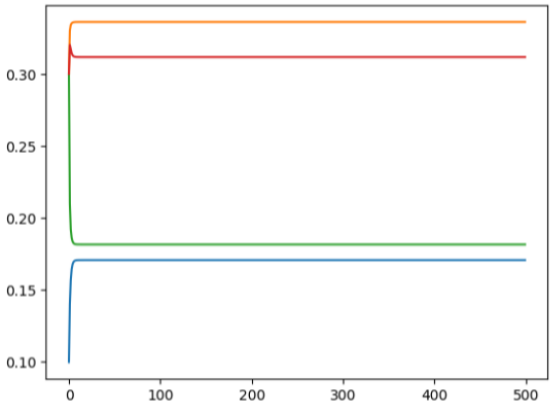
```
[[0.17073171] [0.33604336] [0.18157182] [0.31165312]]
```



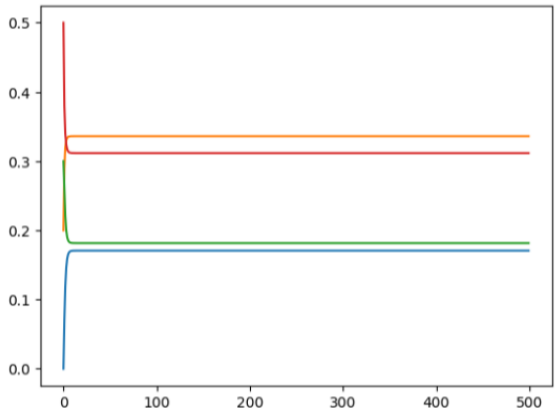
Initial State: Read



Initial State: Write



Initial State: Email



Initial State: Surf

The above graphs for very large values of time(minutes) shows that the probabilities converge after a point in time. Thus, we see a limiting distribution.

Q2. **1-D Random Walk**

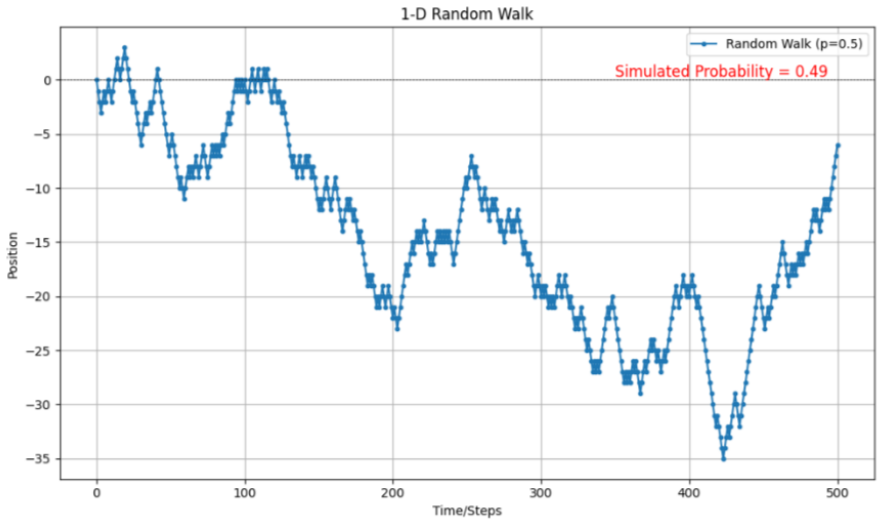
Here we are simulating a simple 1-D Random walk where an object randomly moves along the x-axis either towards the right or left with a probability of ‘p’ and ‘1-p’ respectively.

The code takes the number of steps, probability ‘p’ of transitioning from 1 state to the other & the initial state as input. The function ‘rand_walk’ simulates the random walk for the above parameters using the random function and stores the position after each step in a list for plotting the results. The code also calculates the simulated probability in the random walk of reaching the final position in the simulated random walk.

Code:

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

# Function for simulating a random walk
def rand_walk(steps, p, initial_state):
    pos = [initial_state]
    for i in range(1, steps + 1):
        if random.random() < p:
            pos.append(pos[-1] + 1)
        else:
            pos.append(pos[-1] - 1)
    return pos

# Parameters
steps = 500
p = 0.5
initial_state = 0

pos = rand_walk(steps, p, initial_state)

# Plotting the results
plt.figure(figsize=(10, 6))
plt.plot(pos, linestyle='-', marker='o', markersize=3, label=f'Random Walk (p={p})')
plt.xlabel('Time/Steps')
plt.ylabel('Position')
plt.title('1-D Random Walk')
plt.grid(True)
plt.legend()
plt.axhline(0, color='black', linewidth=0.5, linestyle='--')
plt.tight_layout()

# Calculate the simulated probability
tot = 2 * steps
fin_pos = pos[-1]
dist = fin_pos - (-1 * steps)
sim_prob = dist / tot

plt.annotate(f'Simulated Probability = {sim_prob:.2f}', xy=(0.7 * steps, 0.1 * max(pos)), fontsize=12, color='red')
plt.show()
```

a) Plots:



The simulated probability in the above 1-D Random walk simulation with p = 0.5 , and steps as 500 is **0.49.**

Observations

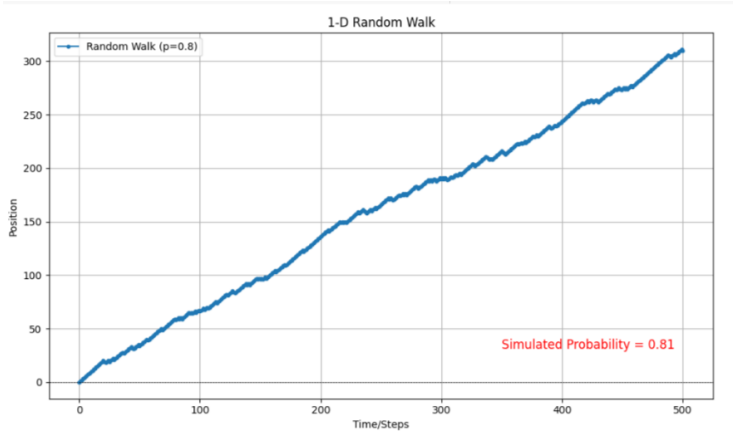
- On increasing the number of steps the simulated probability is closer to p = 0.5.
- The result will vary slightly for each run of the simulation due to the randomness involved in the random walk.

b) Modifying the value of p = 0.8 in the next simulation.

Code:

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

# Function for simulating a random walk
def rand_walk(steps, p, initial_state):
    pos = [initial_state]
    for i in range(1, steps + 1):
        if random.random() < p:
            pos.append(pos[-1] + 1)
        else:
            pos.append(pos[-1] - 1)
    return pos

# Parameters
steps = 500
p = 0.8
initial_state = 0

pos = rand_walk(steps, p, initial_state)

# Plotting the results
plt.figure(figsize=(10, 6))
plt.plot(pos, linestyle='-', marker='o', markersize=3, label=f'Random Walk (p={p})')
plt.xlabel('Time/Steps')
plt.ylabel('Position')
plt.title('1-D Random Walk')
plt.grid(True)
plt.legend()
plt.axhline(0, color='black', linewidth=0.5, linestyle='--')
plt.tight_layout()

# Calculate the simulated probability
tot = 2 * steps
fin_pos = pos[-1]
dist = fin_pos - ((initial_state-steps))
sim_prob = dist / tot

plt.annotate(f'Simulated Probability = {sim_prob:.2f}', xy=(0.7 * steps, 0.1 * max(pos)), fontsize=12, color='red')
plt.show()
```

Plots



The simulated probability in the above 1-D Random walk simulation with p = 0.8 , and steps as 500 is **0.81.**

Observations & Comparisons

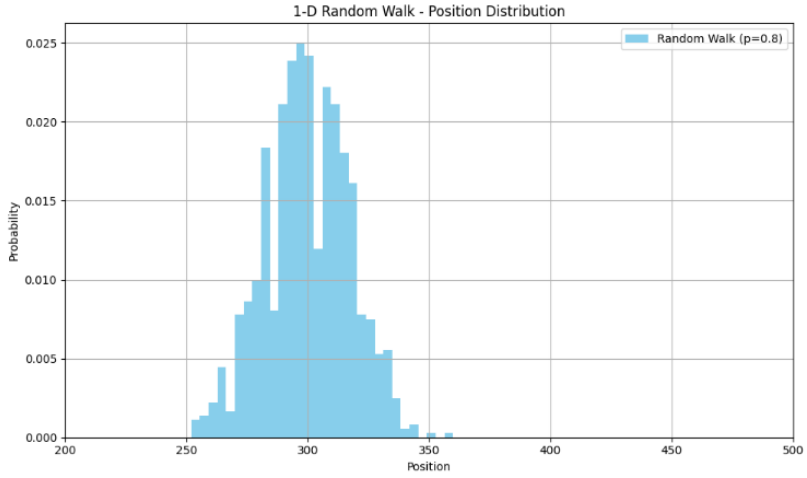
- On changing the value of probability p from 0.5 to 0.8 we can see a significant difference in the behavior of the random walk which can be easily observed from the above obtained plots.
- When p = 0.5 there is an equal probability of moving towards the left or right i.e an unbiased walk, however, with p = 0.8 there is a biased probability of moving rightwards.
- We can see that the random walk drifts in the rightward direction since the position of the object is increasing with time unlike the unbiased walk where one could move either leftward or rightward with increasing time depending on the randomness.

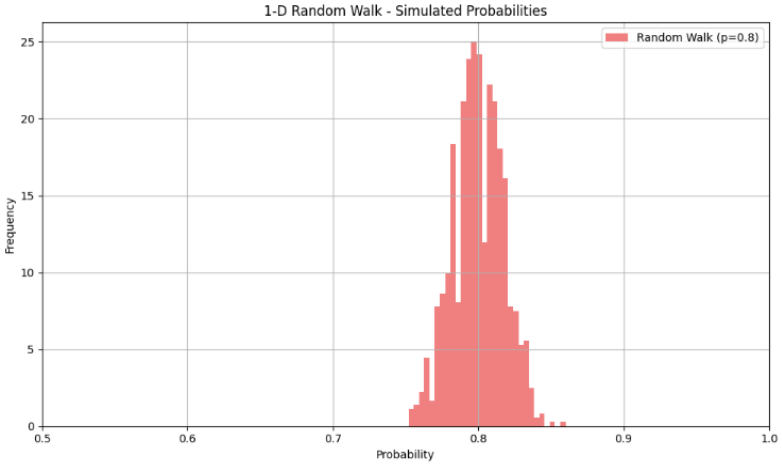
c) Code:

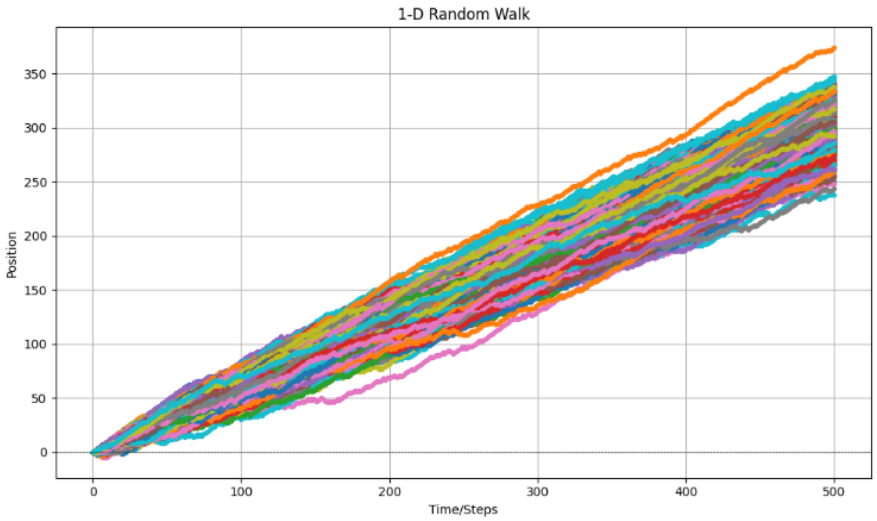
```
import numpy as np
import matplotlib.pyplot as plt
import random

# Function for simulating a random walk
def rand_walk(steps, p, initial_state):
    pos = [initial_state]
    for i in range(1, steps + 1):
        if random.random() < p:
            pos.append(pos[-1] + 1)
        else:
            pos.append(pos[-1] - 1)
    return pos

steps = 500
p = 0.8
initial_state = 0
trials = 1000

# Simulating the random walk
pos = []
simu_prob = []
for i in range(trials):
    pos.append(rand_walk(steps, p, initial_state)[-1])
    simu_prob.append((pos[-1] - (initial_state - steps)) / (2 * steps))

# Plotting the results - Position Distribution
plt.figure(figsize=(10, 6))
plt.hist(pos, bins=30, density=True, label=f'Random Walk (p={p})', color='skyblue')
plt.xlabel('Position')
plt.ylabel('Probability')
plt.title('1-D Random Walk - Position Distribution')
plt.grid(True)
plt.legend()
plt.axvline(0, color='black', linewidth=0.5, linestyle='--')
plt.xlim(200, 500)
plt.tight_layout()
plt.show()

# Plotting the graph of simulated probabilities
plt.figure(figsize=(10, 6))
plt.hist(simu_prob, bins=30, density=True, label=f'Random Walk (p={p})', color='lightcoral')
plt.xlabel('Probability')
plt.ylabel('Frequency')
plt.title('1-D Random Walk - Simulated Probabilities')
plt.grid(True)
plt.legend()
plt.axvline(0, color='black', linewidth=0.5, linestyle='--')
plt.xlim(0.5, 1)
plt.tight_layout()
plt.show()
```

In this part we simply simulate the random walk for 1000 trials in a for loop. In order to make the appropriate comparisons we store the final position after each simulation and similarly the simulated probabilities of each simulation.

Plots:


1-D Random Walk - Position Distribution
Random Walk (p=0.8)
Probability
Position
0.025
0.020
0.015
0.010
0.005
0.000
200
250
300
350
400
450
500


1-D Random Walk - Simulated Probabilities
Random Walk (p=0.8)
Frequency
Probability
25
20
15
10
5
0
0.5
0.6
0.7
0.8
0.9
1.0


1-D Random Walk
Position
Time/Steps
350
300
250
200
150
100
50
0
0
100
200
300
400
500

Observations:

```
#mean and median of final positions & simulated probabilities
print(f'Mean of final positions = {np.mean(pos):.2f}')
print(f'Median of final positions = {np.median(pos):.2f}')
print(f'Mean of simulated probabilities = {np.mean(simu_prob):.2f}')
print(f'Median of simulated probabilities = {np.median(simu_prob):.2f}')
```

```
Mean of final positions = 300.02
Median of final positions = 300.00
Mean of simulated probabilities = 0.80
Median of simulated probabilities = 0.80
```

- The above snippets show the distribution of simulated probabilities and final positions of the objects across the 1000 trials.
- We can see that for a single simulation of the random walk the simulated probabilities are only close to 0.8, however for 1000 trials the simulated probabilities become exactly 0.8.
- Similarly, the final positions in each of the simulations show a rightward movement in the random walk as expected. This can be verified from the graphs as well as the mead/median final positions obtained for 1000 trials.