Math 4430

Final Project

Group

Jay Han

Ahmed Diabate

Zijiang Yan

Yufan Tao

December 2020

Introduction

Monte Carlo Markov Chain (MCMC) is a method used in statistics and statistical physics to extract sequences of random samples from a probability distribution when direct sampling is difficult. The resulting sequence can be used to estimate the probability distribution or calculate integrals (such as expected values).

MCMC will be used in this report to find a solution to a version of the Travelling Salesman Problem. The Travelling Salesman Problem describes a situation where a salesman must travel to a certain number of cities (in this case 20) without visiting a city more than once. Each of the 20 cities are connected to each other, except for 5 pairs of cities, where travel is forbidden from one city i to another city j. The above circuit where one starts at a city, travels to every other city once, and ends up back at the initial city is also known as a Hamiltonian circuit. Usually, with the Travelling Salesman Problem, the objective is to find the shortest distance that can be travelled in total. However, the problem that is given is to find the average travel time between the 20 cities, or the expected travel time for a circuit chosen from all circuits.

Methodology

The map of cities is described as follows:

- The 20 cities are labelled 1, 2, …, 19, 20.
- Then, five pairs of cities labelled as {i, j} are chosen to have the travel between one city to another city forbidden, meaning that the salesman cannot travel from city i to city j.
    - Note – the assumption when it comes to the pair of cities is that travel is forbidden strictly from city i to city j only, and the pair {i, j} does not work in the converse scenario, meaning travel from city j to city i is not restricted.
- Next, the travel distances between every other pair of cities (185 total) are generated randomly from some distribution on the positive reals that has a density.
    - Note – there are 185 distances that need to be generated. Usually when finding how many paths of travel there are for 20 cities if each city can reach every other city, the number would be $\binom{20}{2}$ = 190 pairs. However, we labelled 5 paths of travel as forbidden, thus removing 5 possible paths and leaving us with 185 travel paths, or 185 pairs of cities.
- Using the metropolis algorithm and taking μ to = 1 (which means we never reject the metropolis algorithm and always jump to the next state), we choose large n and m and sample uniformly from the circuits. Afterwards, we are able to compute the average travel time from this sample.

Before coding, we were tasked with three tasks to answer, two of which related to the metropolis algorithm itself, and one that related to the Q transition matrix of the Travelling Salesman Problem. These will be shown first, followed by the results we have observed from coding the Travelling

Salesman Problem along with the problems we faced.

At the end of this report, the coding used through Python is provided through the Appendix.

Task 1

Write out a proof that P is reversible with respect to μ. Conclude that μ is an invariant probability distribution for P.

Let U be a discrete space continuous time and let μ be a target distribution with all $\mu_i > 0$.

Also let Q be an irreducible transition matrix that is symmetric ($q_{ij} = q_{ji}$) for each i, j.

Also, we have:

$P_{ij} = q_{ij} * \left(1, \frac{\mu_j}{\mu_i}\right) \qquad j \neq i$

and

$P_{ij} = 1 - \sum_{k \neq i} q_{ik} * \left(1, \frac{\mu_k}{\mu_i}\right), \qquad j = i$

Recall: in a discrete space U, P is reversible with respect to μ if for all i, j: $\mu_i p_{ij} = \mu_j p_{ji}$

We know $\mu_i = \frac{Pi}{\sum_{i=1}^{\infty} Pi}$ where $\sum_{i=1}^{\infty} p_i < \infty$

and similarly, $\mu_j = \frac{Pj}{\sum_{i=1}^{\infty} Pj}$ where $\sum_{i=1}^{\infty} p_j < \infty$

Thus, from what we said above, that

P is reversible with respect to μ if for all i, j: $\mu_i p_{ij} = \mu_j p_{ji}$.

Given P_{ij}, we´ve got 2 cases here: when i = j (case 1) and when i ≠ j (case 2)

Let´s check if the reversibility work for all cases:

<u>Case 1.1:</u> $\frac{\mu_k}{\mu_i} < 1$

$\mu_i p_{ij} = \mu_i * (1 - \sum_{k \neq i} q_{ik} * \frac{\mu_k}{\mu_i}))$

and $\mu_j p_{ji} = \mu_{j*} * (1 - \sum_{k \neq j} q_{jk} * \frac{\mu_k}{\mu_j}))$

As i = j,

$\mu_j p_{ji} = \mu_i * (1 - \sum_{k \neq i} q_{ik} * \frac{\mu_k}{\mu_i})$

Therefore, reversibility works under this condition.

<u>Case 1.2:</u> $\frac{\mu_k}{\mu_i} \geq 1$

$\mu_i p_{ij} = \mu_i * (1 - \sum_{k \neq i} q_{ik*(1)})) = \mu_i p_{ij} = \mu_i * (1 - \sum_{k \neq i} q_{ik})$

and $\mu_j p_{ji} = \mu_j * (1 - \sum_{k \neq i} q_{jk} * 1)) = \mu_j * (1 - \sum_{k \neq j} q_{jk})$

Again, as i = j,

$\mu_j p_{ji} = \mu_j * (1 - \sum_{k \neq i} q_{ik})) = \mu_j * (1 - \sum_{k \neq j} q_{ik})$

Thus, reversibility also works under this condition.

What is left to show that i ≠ j,

<u>Case 2:</u> i ≠ j,

In this case, then the same condition applies, meaning when $\mu_i p_{ij} = \mu_j p_{ji}$.

In other words,

If μ_i*(q_{ij}*min (1, $\frac{\mu_j}{\mu_i}$)) = μ_j*(q_{ji}*min (1, $\frac{\mu_i}{\mu_j}$))

Case 2.1: $\frac{\mu_j}{\mu_i}=1$

Then,

$\mu_i p_{ij} = \mu_i (q_{ij} *1) = \mu_i * q_{ij}$

$\mu_j p_{ji} = \mu_j *(q_{ji}*1) = \mu_j *(q_{ji}*1) = \mu_i * q_{ij}$

Therefore, it is reversible under this condition.

Case 2.2: $\frac{\mu_j}{\mu_i} > 1, \frac{\mu_i}{\mu_j} < 1$

Then,

$\mu_i p_{ij} = \mu_i (q_{ji}*(1)) = \mu_i q_{ij}$

$\mu_j p_{ji} = \mu_j *(q_{ji}*(1)) = \mu_j *(q_{ji}* \frac{\mu_i}{\mu_j}) = \mu_i q_{ij}$ as Q is symmetric.

$\mu_j p_{ji} = \mu_i p_{ij}$

Therefore, it is reversible under this condition too.

Case 2.3: $\frac{\mu_j}{\mu_i} < 1, \frac{\mu_i}{\mu_j} > 1$

Then,

$\mu_i p_{ij} = \mu_i q_{ij} \frac{\mu_j}{\mu_i} = \mu_j q_{ij}$ and

$\mu_j p_{ji} = \mu_j (q_{ji}*(1)) = \mu_j q_{ji} = \mu_i q_{ij} = \mu_i p_{ij}$

Therefore, it is reversible under this condition.

We have shown in all cases that P is reversible with respect to μ.

Hence by this fact, this implies that μ is an invariant probability distribution with respect to P from what we have seen in class concerning the reversibility topic.

Task 2

Now that we have proven the first task, the second task asks us to prove P is irreducible and P is aperiodic when μ is not perfectly uniform.

From the question, we have known that Q is an irreducible transition matrix.

By the definition of irreducibility, for $\forall\, i, j, i \to j$,which means i communicates with j under Q.

We know $P_{ij} =$
$$\begin{cases} q_{ij} \min(1, \mu_j/\mu_i) & , i \neq j \\ 1 - \sum_{k\neq i} q_{ik} \min(1, \mu_j/\mu_i) & , i = j \end{cases}$$

Since Q is irreducible, then $q_{ij}{}^{(n)} > 0$ for some $n \in Z^+$.

- In the case that $i \neq j$ if $q_{ij}{}^{(n)} > 0$ $P_{ij} = q_{ij} \min(1, \mu_j/\mu_i) > 0$ where $\min(1, \mu_j/\mu_i) > 0$
- In the case that $i = j$ if $q_{ij}{}^{(n)} > 0$, we have
$$0 < \sum_{k\neq i} q_{ik} \min(1, \mu_j/\mu_i) < 1$$
$$P_{ij} = 1 - \sum_{k\neq i} q_{ik} \min(1, \mu_j/\mu_i) > 0$$

Hence, $P_{ij}{}^{n} > 0$ for all i, j and P is irreducible, which means $i \to j$ under P.

Since we know that P is irreducible, there is a single equivalence class which means a state can return to itself with one step.

- Suppose μ is perfectly uniform:
  Then $\mu_i = \frac{1}{\sum i}$ $(i \epsilon S)$

- Suppose μ is not perfectly uniform:
  There exists $\mu_i \neq \mu_j$
  For some i and j, without loss of generality, suppose $\mu_i > \mu_j$
  For the transition from i to j, there is a probability of $p^* = 1 - \frac{\mu_j}{\mu_i}$, where the chain will reject the move from i to j .
  Since $\mu_i > \mu_j$, $p^* = 1 - \frac{\mu_j}{\mu_i} > 0$
  $P_{ii} \geq p^* > 0$
  This period of chain is 1, which means the p is aperiodic.

Task 3

For the Travelling Salesman Problem, task three asks us to check that Q is symmetric.

Q is as follows:

- Let the state space of Q be all the sequences of 20 cities without repetition. Note that some sequences are not valid, as they contain "forbidden trips".
- If i is a sequence above, choose two cities at random, and swap their positions.
  - If the swap creates a sequence with forbidden travel, you would reject the change, and Q will leave the sequence as is.

Note that since the state space of Q is all the sequences of 20 cities (where the cities are labelled 1-20), it can be written as approximately {20!}, and it is close to impossible to write out a matrix that is 20! x 20!. Therefore, we need to write out a written argument that will show that Q can be symmetric.

Consider the following case:

- State i = {1, 2, 3, …, 18, 19, 20}
- State j = {2, 1, 3, …, 18, 19, 20}

Since there is a difference of 2 cities within each state, the probability of moving from state i to state j is $\frac{1}{\binom{20}{2}}$, where $\binom{20}{2}$ =190 is the number of combinations of two cities swapping places.

This means that $q_{ij} = \frac{1}{\binom{20}{2}}$

Since the number of possibilities of choosing two cities to swap is $\binom{20}{2}$, this would also work if we were to move from state j to state i; there would be the same number of possibilities of swapping the position of 2 cities, and the same probability to move back to state i of $\frac{1}{\binom{20}{2}}$. This means that $q_{ji} = \frac{1}{\binom{20}{2}}$ as well, and thus $q_{ij} = q_{ji}$.

If between state i and state j there is a difference of cities in the sequence that is not two (notably anything above two, since you cannot have one difference), then q_{ij} = 0, since with a single swap of only two cities you could not create two separate sequences that have cities in three or more different positions.

This would also mean that q_{ji} is also 0 with the same logic, since going from state j to state i we would have the same problem of not being able to change the positions of three or more cities with a single swap of two cities.

WLOG, this would happen for all states i and j.

Therefore, since the only possibilities for q_{ij} are 0 or $\frac{1}{\binom{20}{2}}$, and the same probability will happen at q_{ji}, as shown above. This proves that Q is a symmetric matrix.

Task 4

The code is in the appendix to this file.

Math 4430 Project Python Code Zijiang Yan

December 4, 2020

[1]:
```python
import numpy as np
import random
from __future__ import division


Q_1 = np.zeros((20,20))


for i in range(0, 20, 1):
    for j in range(0, 20, 1):

            Q_1[i][j]=(1/20)

#print(Q_1)
print("There are 5 pairs forbidden cities")
for k in range(0,5,1):
    value1 = random.randint(0, 20)  #random pick 1 city in the city_list.
    value2 = random.randint(0, 20)  #random pick 1 city in the city_list.
    # we need to make sure different city
    while(value1 == value2):
        value2 = random.randint(0, 20)
    print("The forbiden pair of cities index ", k, " is ",value1,value2)
    for i in range(0, 20, 1):
        for j in range(0, 20, 1):

            if(i == j):
                Q_1[i][j] = 0 # It is not possible if we departure and arrive in␣
 ↪the same city for single route



            #  otherwise, forbidden city pairs should be mark as 0.
        Q_1[value1-1][value2-1] = 0 # value1 to value2 is forbidden
        Q_1[value2-1][value1-1] = 0 #the counterpart is forbidden
#print(Q_1)

def count_non_zero(row):
```

```
    # if transition state is not 0 , means it has the probability to approach␣
↪the other city
    # for example {0.5,0.5,0,0,0,....} the result should be 2
    count= 0
    for j in range(0, 20, 1):
        if  Q_1[row][j] != 0:
            count = count +1

    return count # count is non-zero result

for i in range(0, 20, 1):
    non_zero_count = count_non_zero(i)
    for j in range(0, 20, 1):
        if  Q_1[i][j] != 0: # if there is not in the same city
                Q_1[i][j] = 1/non_zero_count # we redefine the Q matrix make␣
↪non-zero probability are uniform.


print(Q_1)  # old transition matrix
Q =Q_1
```

```
There are 5 pairs forbidden cities
The forbiden pair of cities index  0  is  6 14
The forbiden pair of cities index  1  is  16 13
The forbiden pair of cities index  2  is  5 13
The forbiden pair of cities index  3  is  4 10
The forbiden pair of cities index  4  is  4 9
[[0.         0.05263158 0.05263158 0.05263158 0.05263158 0.05263158
  0.05263158 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158
  0.05263158 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158
  0.05263158 0.05263158]
 [0.05263158 0.         0.05263158 0.05263158 0.05263158 0.05263158
  0.05263158 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158
  0.05263158 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158
  0.05263158 0.05263158]
 [0.05263158 0.05263158 0.         0.05263158 0.05263158 0.05263158
  0.05263158 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158
  0.05263158 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158
  0.05263158 0.05263158]
 [0.05882353 0.05882353 0.05882353 0.         0.05882353 0.05882353
  0.05882353 0.05882353 0.         0.         0.05882353 0.05882353
  0.05882353 0.05882353 0.05882353 0.05882353 0.05882353 0.05882353
  0.05882353 0.05882353]
 [0.05555556 0.05555556 0.05555556 0.05555556 0.         0.05555556
  0.05555556 0.05555556 0.05555556 0.05555556 0.05555556 0.05555556
  0.         0.05555556 0.05555556 0.05555556 0.05555556 0.05555556
  0.05555556 0.05555556]
```

```
[0.05555556 0.05555556 0.05555556 0.05555556 0.05555556 0.
 0.05555556 0.05555556 0.05555556 0.05555556 0.05555556 0.05555556
 0.05555556 0.         0.05555556 0.05555556 0.05555556 0.05555556
 0.05555556 0.05555556]
[0.05263158 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158
 0.         0.05263158 0.05263158 0.05263158 0.05263158 0.05263158
 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158
 0.05263158 0.05263158]
[0.05263158 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158
 0.05263158 0.         0.05263158 0.05263158 0.05263158 0.05263158
 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158
 0.05263158 0.05263158]
[0.05555556 0.05555556 0.05555556 0.         0.05555556 0.05555556
 0.05555556 0.05555556 0.         0.05555556 0.05555556 0.05555556
 0.05555556 0.05555556 0.05555556 0.05555556 0.05555556 0.05555556
 0.05555556 0.05555556]
[0.05555556 0.05555556 0.05555556 0.         0.05555556 0.05555556
 0.05555556 0.05555556 0.05555556 0.         0.05555556 0.05555556
 0.05555556 0.05555556 0.05555556 0.05555556 0.05555556 0.05555556
 0.05555556 0.05555556]
[0.05263158 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158
 0.05263158 0.05263158 0.05263158 0.05263158 0.         0.05263158
 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158
 0.05263158 0.05263158]
[0.05263158 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158
 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158 0.
 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158
 0.05263158 0.05263158]
[0.05882353 0.05882353 0.05882353 0.05882353 0.         0.05882353
 0.05882353 0.05882353 0.05882353 0.05882353 0.05882353 0.05882353
 0.         0.05882353 0.05882353 0.         0.05882353 0.05882353
 0.05882353 0.05882353]
[0.05555556 0.05555556 0.05555556 0.05555556 0.05555556 0.
 0.05555556 0.05555556 0.05555556 0.05555556 0.05555556 0.05555556
 0.05555556 0.         0.05555556 0.05555556 0.05555556 0.05555556
 0.05555556 0.05555556]
[0.05263158 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158
 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158
 0.05263158 0.05263158 0.         0.05263158 0.05263158 0.05263158
 0.05263158 0.05263158]
[0.05555556 0.05555556 0.05555556 0.05555556 0.05555556 0.05555556
 0.05555556 0.05555556 0.05555556 0.05555556 0.05555556 0.05555556
 0.         0.05555556 0.05555556 0.         0.05555556 0.05555556
 0.05555556 0.05555556]
[0.05263158 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158
 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158
 0.05263158 0.05263158 0.05263158 0.05263158 0.         0.05263158
 0.05263158 0.05263158]
```

```
[0.05263158 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158
 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158
 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158 0.
 0.05263158 0.05263158]
[0.05263158 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158
 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158
 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158
 0.         0.05263158]
[0.05263158 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158
 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158
 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158 0.05263158
 0.05263158 0.        ]]
```

[2]:

```
Q_2 = np.zeros((20,20)) #travel distance matrix
for i in range(0, 20, 1):
    for j in range(0, 20, 1):
        # 1/20 no forbidde
            if(i == j):
                Q_2[i][j] = 0 # depature and arrival are same city ,mark is as 0

            else:
                value = random.randint(1, 100)  #find random distance from 0 to
 ↪100.
                Q_2[i][j] = value # assign the distance to the i,j entry.

#print(Q_2)

for i in range(0, 20, 1):
    for j in range(0, 20, 1):
        if Q_2[i][j] != 0:
            Q_2[j][i] = Q_2[i][j] #  we need to make sure the distance from city
 ↪A to city B is the same as the city B to city A
        #make sure it is symmetry
print(Q_2)
```

```
[[ 0. 44. 10. 63. 31. 65. 30. 59.  4. 90. 94. 99. 78. 76.
  21. 45. 23. 27. 19. 13.]
 [ 44.  0. 19. 83. 37. 90. 59. 90. 76. 86. 29. 64. 58. 53.
  92. 58. 33. 40. 84. 71.]
 [ 10. 19.  0. 92. 47. 33. 89. 18. 31. 98. 24. 78. 20.  5.
  17. 54. 80. 60. 32. 25.]
 [ 63. 83. 92.  0. 35. 12. 20. 29. 57. 21. 47. 55. 93. 55.
  30. 23. 30. 54. 96. 34.]
 [ 31. 37. 47. 35.  0. 99. 76. 53. 72. 96. 27. 12. 55. 38.
  27. 78. 40. 38. 35. 30.]
 [ 65. 90. 33. 12. 99.  0. 48. 83.  5. 17. 50. 52. 81. 11.
   7. 71. 49. 80.  8. 60.]
 [ 30. 59. 89. 20. 76. 48.  0. 18. 85. 40. 45. 59. 78. 74.
```

```
  63.  46.  91.  96.  46.  87.]
 [ 59.  90.  18.  29.  53.  83.  18.   0.  54.  21.  33.  31.  92.  39.
  73.  25.  11.  98.  20.  35.]
 [  4.  76.  31.  57.  72.   5.  85.  54.   0.  22.   7.  41.  40. 100.
  70.  33.  42.  77.  82.  86.]
 [ 90.  86.  98.  21.  96.  17.  40.  21.  22.   0.  98.  98.  34.  80.
  85.  18.  81.  30.   8.  55.]
 [ 94.  29.  24.  47.  27.  50.  45.  33.   7.  98.   0.  80.  83.  72.
  88.  22.  84.  85.  71.  57.]
 [ 99.  64.  78.  55.  12.  52.  59.  31.  41.  98.  80.   0.  75.  70.
  94.  85.  33.  96.   6.  82.]
 [ 78.  58.  20.  93.  55.  81.  78.  92.  40.  34.  83.  75.   0.  37.
  92.  22.  73.  84.  86.  33.]
 [ 76.  53.   5.  55.  38.  11.  74.  39. 100.  80.  72.  70.  37.   0.
  18.  83.  59.  56.  94.  65.]
 [ 21.  92.  17.  30.  27.   7.  63.  73.  70.  85.  88.  94.  92.  18.
   0.   6.  95.  16.  31.  35.]
 [ 45.  58.  54.  23.  78.  71.  46.  25.  33.  18.  22.  85.  22.  83.
   6.   0.  89.  65.   4.  50.]
 [ 23.  33.  80.  30.  40.  49.  91.  11.  42.  81.  84.  33.  73.  59.
  95.  89.   0.   5.   6.  29.]
 [ 27.  40.  60.  54.  38.  80.  96.  98.  77.  30.  85.  96.  84.  56.
  16.  65.   5.   0.  88.  41.]
 [ 19.  84.  32.  96.  35.   8.  46.  20.  82.   8.  71.   6.  86.  94.
  31.   4.   6.  88.   0.  99.]
 [ 13.  71.  25.  34.  30.  60.  87.  35.  86.  55.  57.  82.  33.  65.
  35.  50.  29.  41.  99.   0.]]
```

[3]:
```python
# Metropolis Algorithm
pi=[] # we construct a uniform initial distributution
for j in range(0, 20, 1):
        # 1/20 no forbidde
        pi.append(1/20)
print("Initial Distribution ",pi)
```

```
Initial Distribution  [0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05,
0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05]
```

[4]:
```python
from __future__ import division
import math

# this part is simply compute  alpha = min(1, miu_k/miu_j)
def alpha (r,c):#r,c from 0-4
    r = r-1
    c = c-1
    if((pi[r]*Q[r][c])==0):
        x = -1 #Denominator is 0 error
    else:
```

```
        x = (pi[c]*Q[c][r]* 1.0)/(pi[r]*Q[r][c])

    if(x>1):
        result = 1
    elif (x > 0): # 0<x<1
        result = x
    else:
        result = -1 #error
    return result
```

[5]:
```
def new_transition(i,j):#compute  new transition matrix
    i = i-1
    j = j-1
    result  =-1 #error
    #if r != c:
    if alpha(i,j) != -1: #if i != j:
        result = Q[i][j] * alpha(i+1,j+1) *1.0
    else: # if i == j
        result  = 1
        for k in range(0, len(Q), 1):
            if k != i:
                result = result - Q[i][k] * alpha(i+1,k+1) *1.0

    return result


for i in range(1, len(pi)+1, 1): #compute diagonal of new transition matrix P
    print("fnewTransition(%d,%d)is%f",i,i,new_transition(i,i))
```

```
fnewTransition(%d,%d)is%f 1 1 5.551115123125783e-16
fnewTransition(%d,%d)is%f 2 2 5.551115123125783e-16
fnewTransition(%d,%d)is%f 3 3 5.551115123125783e-16
fnewTransition(%d,%d)is%f 4 4 0.08737530099759233
fnewTransition(%d,%d)is%f 5 5 0.035087719298245806
fnewTransition(%d,%d)is%f 6 6 0.035087719298245806
fnewTransition(%d,%d)is%f 7 7 5.551115123125783e-16
fnewTransition(%d,%d)is%f 8 8 5.551115123125783e-16
fnewTransition(%d,%d)is%f 9 9 0.03508771929824592
fnewTransition(%d,%d)is%f 10 10 0.03508771929824592
fnewTransition(%d,%d)is%f 11 11 5.551115123125783e-16
fnewTransition(%d,%d)is%f 12 12 5.551115123125783e-16
fnewTransition(%d,%d)is%f 13 13 0.08737530099759222
fnewTransition(%d,%d)is%f 14 14 0.035087719298245806
fnewTransition(%d,%d)is%f 15 15 5.551115123125783e-16
fnewTransition(%d,%d)is%f 16 16 0.03508771929824578
fnewTransition(%d,%d)is%f 17 17 5.551115123125783e-16
fnewTransition(%d,%d)is%f 18 18 5.551115123125783e-16
fnewTransition(%d,%d)is%f 19 19 5.551115123125783e-16
```

```
fnewTransition(%d,%d)is%f 20 20 5.551115123125783e-16
```

[10]:

```python
A = np.zeros((len(Q),len(Q[0])))

for i in range(0, len(Q), 1):
    for j in range(0, len(Q[0]), 1):
            A[i][j] = new_transition(i,j)
display("Initial transition matrix")
print(A)
from numpy.linalg import matrix_power
display("transition matrix Repeating 10 times")
A_2 =matrix_power(A, 10) #This is the new Transition Matrix after 10 repetitions.
print(A_2)

display("transition matrix Repeating 20 times")
A_2 =matrix_power(A, 20) #This is the new Transition Matrix after 10 repetitions.
print(A_2)
#A_100 =matrix_power(A, 100)
#print(A_100)
```

```
'Initial transition matrix'
```

```
[[ 5.55111512e-16  5.26315789e-02  5.26315789e-02  5.26315789e-02
   5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
   5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
   5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
   5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02]
 [ 5.26315789e-02  5.55111512e-16  5.26315789e-02  5.26315789e-02
   5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
   5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
   5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
   5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02]
 [ 5.26315789e-02  5.26315789e-02  5.55111512e-16  5.26315789e-02
   5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
   5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
   5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
   5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02]
 [ 5.26315789e-02  5.26315789e-02  5.26315789e-02  5.55111512e-16
   5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
   5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
   5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
   5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02]
 [ 5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
   8.73753010e-02  5.55555556e-02  5.55555556e-02  5.26315789e-02
   5.26315789e-02 -0.00000000e+00 -0.00000000e+00  5.26315789e-02
   5.26315789e-02  5.88235294e-02  5.55555556e-02  5.26315789e-02
   5.55555556e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02]
 [ 5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
```

```
 5.55555556e-02  3.50877193e-02  5.55555556e-02  5.26315789e-02
 5.26315789e-02  5.55555556e-02  3.50877193e-02  3.50877193e-02
 5.26315789e-02 -0.00000000e+00  5.55555556e-02  5.26315789e-02
 5.55555556e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02]
[ 5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
 5.55555556e-02  5.55555556e-02  3.50877193e-02  5.26315789e-02
 5.26315789e-02  5.55555556e-02  5.55555556e-02  5.26315789e-02
 5.26315789e-02  5.55555556e-02  3.50877193e-02  5.26315789e-02
 5.55555556e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02]
[ 5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
 5.26315789e-02  5.26315789e-02  5.26315789e-02  5.55111512e-16
 5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
 5.26315789e-02  5.26315789e-02  5.26315789e-02  5.55111512e-16
 5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02]
[ 5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
 5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
 5.55111512e-16  5.26315789e-02  5.26315789e-02  5.26315789e-02
 5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
 5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02]
[ 5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
-0.00000000e+00  5.55555556e-02  5.55555556e-02  5.26315789e-02
 5.26315789e-02  3.50877193e-02  5.55555556e-02  5.26315789e-02
 5.26315789e-02  5.55555556e-02  5.55555556e-02  5.26315789e-02
 5.55555556e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02]
[ 5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
-0.00000000e+00  3.50877193e-02  5.55555556e-02  5.26315789e-02
 5.26315789e-02  5.55555556e-02  3.50877193e-02  5.26315789e-02
 5.26315789e-02  5.55555556e-02  5.55555556e-02  5.26315789e-02
 5.55555556e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02]
[ 5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
 5.26315789e-02  5.55111512e-16  5.26315789e-02  5.26315789e-02
 5.26315789e-02  5.26315789e-02  5.26315789e-02  5.55111512e-16
 5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
 5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02]
[ 5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
 5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
 5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
 5.55111512e-16  5.26315789e-02  5.26315789e-02  5.26315789e-02
 5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02]
[ 5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
 5.88235294e-02 -0.00000000e+00  5.55555556e-02  5.26315789e-02
 5.26315789e-02  5.55555556e-02  5.55555556e-02  5.26315789e-02
 5.26315789e-02  8.73753010e-02  5.55555556e-02  5.26315789e-02
-0.00000000e+00  5.26315789e-02  5.26315789e-02  5.26315789e-02]
[ 5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
 5.55555556e-02  5.55555556e-02  3.50877193e-02  5.26315789e-02
 5.26315789e-02  5.55555556e-02  5.55555556e-02  5.26315789e-02
 5.26315789e-02  5.55555556e-02  3.50877193e-02  5.26315789e-02
```

```
  5.55555556e-02  3.50877193e-02  5.26315789e-02  5.26315789e-02]
 [ 5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
  5.26315789e-02  5.26315789e-02  5.26315789e-02  5.55111512e-16
  5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
  5.26315789e-02  5.26315789e-02  5.26315789e-02  5.55111512e-16
  5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02]
 [ 5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
  5.55555556e-02  5.55555556e-02  5.55555556e-02  5.26315789e-02
  5.26315789e-02  5.55555556e-02  5.55555556e-02  5.26315789e-02
  5.26315789e-02 -0.00000000e+00  5.55555556e-02  5.26315789e-02
  3.50877193e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02]
 [ 5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
  5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
  5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
  5.26315789e-02  5.26315789e-02  5.55111512e-16  5.26315789e-02
  5.26315789e-02  5.55111512e-16  5.26315789e-02  5.26315789e-02]
 [ 5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
  5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
  5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
  5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
  5.26315789e-02  5.26315789e-02  5.55111512e-16  5.26315789e-02]
 [ 5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
  5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
  5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
  5.26315789e-02  5.26315789e-02  5.26315789e-02  5.26315789e-02
  5.26315789e-02  5.26315789e-02  5.26315789e-02  5.55111512e-16]]
```

```
'transition matrix Repeating 10 times'
```

```
[[0.04592766 0.04592766 0.04592766 0.04592766 0.04597006 0.04263984
  0.0475301  0.04371989 0.04592766 0.04591616 0.04505213 0.04520991
  0.04592766 0.04610995 0.0451279  0.04371989 0.04590846 0.04516803
  0.04592766 0.04592766]
 [0.04592766 0.04592766 0.04592766 0.04592766 0.04597006 0.04263984
  0.0475301  0.04371989 0.04592766 0.04591616 0.04505213 0.04520991
  0.04592766 0.04610995 0.0451279  0.04371989 0.04590846 0.04516803
  0.04592766 0.04592766]
 [0.04592766 0.04592766 0.04592766 0.04592766 0.04597006 0.04263984
  0.0475301  0.04371989 0.04592766 0.04591616 0.04505213 0.04520991
  0.04592766 0.04610995 0.0451279  0.04371989 0.04590846 0.04516803
  0.04592766 0.04592766]
 [0.04592766 0.04592766 0.04592766 0.04592766 0.04597006 0.04263984
  0.0475301  0.04371989 0.04592766 0.04591616 0.04505213 0.04520991
  0.04592766 0.04610995 0.0451279  0.04371989 0.04590846 0.04516803
  0.04592766 0.04592766]
 [0.0459802  0.0459802  0.0459802  0.0459802  0.04602266 0.04268863
  0.04758448 0.04376991 0.0459802  0.04596869 0.04510367 0.04526164
  0.0459802  0.04616271 0.04517953 0.04376991 0.04596099 0.04521971
```

```
 0.0459802  0.0459802 ]
[0.04425083 0.04425083 0.04425083 0.04425083 0.04429169 0.04108306
 0.04579477 0.04212367 0.04425083 0.04423975 0.04340727 0.04355929
 0.04425083 0.04442647 0.04348027 0.04212367 0.04423234 0.04351894
 0.04425083 0.04425083]
[0.04750658 0.04750658 0.04750658 0.04750658 0.04755044 0.04410573
 0.04916411 0.04522291 0.04750658 0.04749468 0.04660095 0.04676416
 0.04750658 0.04769514 0.04667932 0.04522291 0.04748672 0.04672084
 0.04750658 0.04750658]
[0.04371989 0.04371989 0.04371989 0.04371989 0.04376026 0.04059012
 0.0452453  0.04161825 0.04371989 0.04370894 0.04288645 0.04303665
 0.04371989 0.04389342 0.04295857 0.04161825 0.04370162 0.04299678
 0.04371989 0.04371989]
[0.04592766 0.04592766 0.04592766 0.04592766 0.04597006 0.04263984
 0.0475301  0.04371989 0.04592766 0.04591616 0.04505213 0.04520991
 0.04592766 0.04610995 0.0451279  0.04371989 0.04590846 0.04516803
 0.04592766 0.04592766]
[0.04592462 0.04592462 0.04592462 0.04592462 0.04596703 0.04263703
 0.04752696 0.043717   0.04592462 0.04591313 0.04504915 0.04520693
 0.04592462 0.04610691 0.04512492 0.043717   0.04590543 0.04516505
 0.04592462 0.04592462]
[0.04502795 0.04502795 0.04502795 0.04502795 0.04506952 0.04180454
 0.046599   0.04286343 0.04502795 0.04501668 0.04416957 0.04432427
 0.04502795 0.04520667 0.04424386 0.04286343 0.04500913 0.04428321
 0.04502795 0.04502795]
[0.04369308 0.04369308 0.04369308 0.04369308 0.04373342 0.04056523
 0.04521756 0.04159272 0.04369308 0.04368214 0.04286014 0.04301025
 0.04369308 0.0438665  0.04293223 0.04159272 0.04367482 0.04297041
 0.04369308 0.04369308]
[0.04592766 0.04592766 0.04592766 0.04592766 0.04597006 0.04263984
 0.0475301  0.04371989 0.04592766 0.04591616 0.04505213 0.04520991
 0.04592766 0.04610995 0.0451279  0.04371989 0.04590846 0.04516803
 0.04592766 0.04592766]
[0.04602527 0.04602527 0.04602527 0.04602527 0.04606776 0.04273046
 0.04763112 0.04381281 0.04602527 0.04601374 0.04514788 0.045306
 0.04602527 0.04620795 0.04522381 0.04381281 0.04600603 0.04526403
 0.04602527 0.04602527]
[0.04673421 0.04673421 0.04673421 0.04673421 0.04677736 0.04338866
 0.0483648  0.04448767 0.04673421 0.04672251 0.04584331 0.04600387
 0.04673421 0.04691971 0.04592041 0.04448767 0.04671468 0.04596125
 0.04673421 0.04673421]
[0.04371989 0.04371989 0.04371989 0.04371989 0.04376026 0.04059012
 0.0452453  0.04161825 0.04371989 0.04370894 0.04288645 0.04303665
 0.04371989 0.04389342 0.04295857 0.04161825 0.04370162 0.04299678
 0.04371989 0.04371989]
[0.04592215 0.04592215 0.04592215 0.04592215 0.04596455 0.04263473
 0.0475244  0.04371464 0.04592215 0.04591065 0.04504672 0.04520449
 0.04592215 0.04610442 0.04512248 0.04371464 0.04590295 0.04516261
```

```
 0.04592215 0.04592215]
 [0.04356767 0.04356767 0.04356767 0.04356767 0.0436079  0.0404488
  0.04508777 0.04147335 0.04356767 0.04355676 0.04273713 0.04288681
  0.04356767 0.0437406  0.04280901 0.04147335 0.04354946 0.04284708
  0.04356767 0.04356767]
 [0.04592766 0.04592766 0.04592766 0.04592766 0.04597006 0.04263984
  0.0475301  0.04371989 0.04592766 0.04591616 0.04505213 0.04520991
  0.04592766 0.04610995 0.0451279  0.04371989 0.04590846 0.04516803
  0.04592766 0.04592766]
 [0.04592766 0.04592766 0.04592766 0.04592766 0.04597006 0.04263984
  0.0475301  0.04371989 0.04592766 0.04591616 0.04505213 0.04520991
  0.04592766 0.04610995 0.0451279  0.04371989 0.04590846 0.04516803
  0.04592766 0.04592766]]

'transition matrix Repeating 20 times'


[[0.04137618 0.04137618 0.04137618 0.04137618 0.04141438 0.03841419
  0.04281982 0.0393872  0.04137618 0.04136582 0.04058741 0.04072956
  0.04137618 0.04154041 0.04065567 0.0393872  0.04135889 0.04069183
  0.04137618 0.04137618]
 [0.04137618 0.04137618 0.04137618 0.04137618 0.04141438 0.03841419
  0.04281982 0.0393872  0.04137618 0.04136582 0.04058741 0.04072956
  0.04137618 0.04154041 0.04065567 0.0393872  0.04135889 0.04069183
  0.04137618 0.04137618]
 [0.04137618 0.04137618 0.04137618 0.04137618 0.04141438 0.03841419
  0.04281982 0.0393872  0.04137618 0.04136582 0.04058741 0.04072956
  0.04137618 0.04154041 0.04065567 0.0393872  0.04135889 0.04069183
  0.04137618 0.04137618]
 [0.04137618 0.04137618 0.04137618 0.04137618 0.04141438 0.03841419
  0.04281982 0.0393872  0.04137618 0.04136582 0.04058741 0.04072956
  0.04137618 0.04154041 0.04065567 0.0393872  0.04135889 0.04069183
  0.04137618 0.04137618]
 [0.04142351 0.04142351 0.04142351 0.04142351 0.04146176 0.03845814
  0.04286881 0.03943226 0.04142351 0.04141314 0.04063385 0.04077616
  0.04142351 0.04158793 0.04070219 0.03943226 0.0414062  0.04073839
  0.04142351 0.04142351]
 [0.03986553 0.03986553 0.03986553 0.03986553 0.03990234 0.03701168
  0.04125646 0.03794917 0.03986553 0.03985555 0.03910556 0.03924252
  0.03986553 0.04002376 0.03917133 0.03794917 0.03984887 0.03920617
  0.03986553 0.03986553]
 [0.04279862 0.04279862 0.04279862 0.04279862 0.04283814 0.03973481
  0.04429189 0.04074127 0.04279862 0.04278791 0.04198274 0.04212978
  0.04279862 0.0429685  0.04205335 0.04074127 0.04278074 0.04209075
  0.04279862 0.04279862]
 [0.0393872  0.0393872  0.0393872  0.0393872  0.03942357 0.0365676
  0.04076144 0.03749383 0.0393872  0.03937734 0.03863635 0.03877167
  0.0393872  0.03954354 0.03870133 0.03749383 0.03937074 0.03873575
  0.0393872  0.0393872 ]
```

```
[0.04137618 0.04137618 0.04137618 0.04137618 0.04141438 0.03841419
 0.04281982 0.0393872  0.04137618 0.04136582 0.04058741 0.04072956
 0.04137618 0.04154041 0.04065567 0.0393872  0.04135889 0.04069183
 0.04137618 0.04137618]
[0.04137344 0.04137344 0.04137344 0.04137344 0.04141165 0.03841165
 0.04281699 0.0393846  0.04137344 0.04136309 0.04058473 0.04072687
 0.04137344 0.04153766 0.04065299 0.0393846  0.04135615 0.04068915
 0.04137344 0.04137344]
[0.04056563 0.04056563 0.04056563 0.04056563 0.04060309 0.03766167
 0.04198099 0.03861562 0.04056563 0.04055547 0.03979232 0.03993168
 0.04056563 0.04072664 0.03985924 0.03861562 0.04054868 0.03989469
 0.04056563 0.04056563]
[0.03936305 0.03936305 0.03936305 0.03936305 0.03939939 0.03654517
 0.04073645 0.03747084 0.03936305 0.03935319 0.03861266 0.03874789
 0.03936305 0.03951928 0.0386776  0.03747084 0.0393466  0.038712
 0.03936305 0.03936305]
[0.04137618 0.04137618 0.04137618 0.04137618 0.04141438 0.03841419
 0.04281982 0.0393872  0.04137618 0.04136582 0.04058741 0.04072956
 0.04137618 0.04154041 0.04065567 0.0393872  0.04135889 0.04069183
 0.04137618 0.04137618]
[0.04146411 0.04146411 0.04146411 0.04146411 0.0415024  0.03849583
 0.04291082 0.03947091 0.04146411 0.04145373 0.04067367 0.04081613
 0.04146411 0.04162869 0.04074208 0.03947091 0.04144679 0.04077831
 0.04146411 0.04146411]
[0.0421028  0.0421028  0.0421028  0.0421028  0.04214168 0.0390888
 0.0435718  0.0400789  0.0421028  0.04209226 0.04130019 0.04144483
 0.0421028  0.04226992 0.04136965 0.0400789  0.04208521 0.04140644
 0.0421028  0.0421028 ]
[0.0393872  0.0393872  0.0393872  0.0393872  0.03942357 0.0365676
 0.04076144 0.03749383 0.0393872  0.03937734 0.03863635 0.03877167
 0.0393872  0.03954354 0.03870133 0.03749383 0.03937074 0.03873575
 0.0393872  0.0393872 ]
[0.04137121 0.04137121 0.04137121 0.04137121 0.04140941 0.03840958
 0.04281468 0.03938247 0.04137121 0.04136085 0.04058254 0.04072468
 0.04137121 0.04153542 0.0406508  0.03938247 0.04135392 0.04068695
 0.04137121 0.04137121]
[0.03925007 0.03925007 0.03925007 0.03925007 0.03928631 0.03644028
 0.04061953 0.03736329 0.03925007 0.03924024 0.03850183 0.03863668
 0.03925007 0.03940586 0.03856659 0.03736329 0.03923366 0.03860089
 0.03925007 0.03925007]
[0.04137618 0.04137618 0.04137618 0.04137618 0.04141438 0.03841419
 0.04281982 0.0393872  0.04137618 0.04136582 0.04058741 0.04072956
 0.04137618 0.04154041 0.04065567 0.0393872  0.04135889 0.04069183
 0.04137618 0.04137618]
[0.04137618 0.04137618 0.04137618 0.04137618 0.04141438 0.03841419
 0.04281982 0.0393872  0.04137618 0.04136582 0.04058741 0.04072956
 0.04137618 0.04154041 0.04065567 0.0393872  0.04135889 0.04069183
 0.04137618 0.04137618]]
```

[12]:
```
display(' since all entries are approximately equal to 0.05, then we can prove␣
↪the matrix A is reversible, and the pi[0.05,0.05,....,0.05]is the initial␣
↪distribution')
display(' since all entries are greater than 0 after repeation(10,100␣
↪times),then we can prove the matrix A is irreducible')
```

```
' since all entries are approximately equal to 0.05, then we can prove the matrix A is reversibl

' since all entries are greater than 0 after repeation(10,100 times),then we can prove the matri
```

[14]:
```
#A_2 is new transition unver metropolis algorithm



def travel_time(i):
    travel_1 = 0
    # start from 1 come back to 1
    #for i in range(0, 20, 1):
    for j in range(0, 20, 1):
            #1 transit point  i -> j  -> i
            travel_1 += A_2[i][j] * Q_2[i][j] + A_2[j][i] * Q_2[j][i] # distance␣
↪* probability

    return travel_1



def travel_time_2(i):
    distance_2 = 0
    for j in range(0, 20, 1):
             for k in range(0, 20, 1):
                 # 2 transit point  i -> j -> k -> i
                     distance_2 += A_2[i][j] * Q_2[i][j] + A_2[j][k] * Q_2[j][k]␣
↪+ A_2[k][i] * Q_2[k][i] # distance * probability
    return distance_2

for i in range(0, 20, 1):
    print("the 1 iteration average time of city ", i," is ",travel_time(i) )
    print("the 2 iteration average time of city ", i," is ",travel_time_2(i) )
    print(travel_time)
```

```
the 1 iteration average time of city  0  is  72.6446240748816
the 2 iteration average time of city  0  is  2272.7862271537856
<function travel_time at 0x0000028421DA9280>
the 1 iteration average time of city  1  is  95.35328032721448
the 2 iteration average time of city  1  is  2726.9593522004375
```

```
<function travel_time at 0x0000028421DA9280>
the 1 iteration average time of city  2  is  68.2588816778353
the 2 iteration average time of city  2  is  2185.071379212859
<function travel_time at 0x0000028421DA9280>
the 1 iteration average time of city  3  is  76.31661958260571
the 2 iteration average time of city  3  is  2346.2261373082656
<function travel_time at 0x0000028421DA9280>
the 1 iteration average time of city  4  is  75.78655121649018
the 2 iteration average time of city  4  is  2335.6247699859578
<function travel_time at 0x0000028421DA9280>
the 1 iteration average time of city  5  is  71.23964828655951
the 2 iteration average time of city  5  is  2244.686711387343
<function travel_time at 0x0000028421DA9280>
the 1 iteration average time of city  6  is  97.48348762776247
the 2 iteration average time of city  6  is  2769.5634982114016
<function travel_time at 0x0000028421DA9280>
the 1 iteration average time of city  7  is  68.70277319958304
the 2 iteration average time of city  7  is  2193.9492096478143
<function travel_time at 0x0000028421DA9280>
the 1 iteration average time of city  8  is  80.99742037612431
the 2 iteration average time of city  8  is  2439.842153178639
<function travel_time at 0x0000028421DA9280>
the 1 iteration average time of city  9  is  88.6018916588495
the 2 iteration average time of city  9  is  2591.931578833141
<function travel_time at 0x0000028421DA9280>
the 1 iteration average time of city  10  is  88.21090683802635
the 2 iteration average time of city  10  is  2584.111882416681
<function travel_time at 0x0000028421DA9280>
the 1 iteration average time of city  11  is  96.03681534846473
the 2 iteration average time of city  11  is  2740.6300526254468
<function travel_time at 0x0000028421DA9280>
the 1 iteration average time of city  12  is  99.31336885362722
the 2 iteration average time of city  12  is  2806.1611227287003
<function travel_time at 0x0000028421DA9280>
the 1 iteration average time of city  13  is  89.27726524823913
the 2 iteration average time of city  13  is  2605.4390506209356
<function travel_time at 0x0000028421DA9280>
the 1 iteration average time of city  14  is  78.85350532408285
the 2 iteration average time of city  14  is  2396.9638521378106
<function travel_time at 0x0000028421DA9280>
the 1 iteration average time of city  15  is  68.41528893955049
the 2 iteration average time of city  15  is  2188.1995244471645
<function travel_time at 0x0000028421DA9280>
the 1 iteration average time of city  16  is  78.26742924471891
the 2 iteration average time of city  16  is  2385.24233055053
<function travel_time at 0x0000028421DA9280>
the 1 iteration average time of city  17  is  89.74616552786652
the 2 iteration average time of city  17  is  2614.817056213483
```

```
<function travel_time at 0x0000028421DA9280>
the 1 iteration average time of city  18  is  75.36881176536473
the 2 iteration average time of city  18  is  2327.2699809634473
<function travel_time at 0x0000028421DA9280>
the 1 iteration average time of city  19  is  80.9127561944581
the 2 iteration average time of city  19  is  2438.1488695453145
<function travel_time at 0x0000028421DA9280>
```

[9]:
```
def check_symmetric(a, rtol=1e-05, atol=1e-08): # check Q is symmetry
    return not np.allclose(a, a.T, rtol=rtol, atol=atol)

check_symmetric(A)
print("Check new Transition Matrix A is symmetry: ",check_symmetric(A))
display(We have prove this  travelling matrix is symmetry )
```

```
Check new Transition Matrix A is symmetry:  True
```

[ ]:
```

```