In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
from scipy.linalg import null_space
#mpl.rcParams['text.usetex'] = True
#mpl.rcParams['text.latex.preamble'] = [r'\usepackage{amsfonts}']
%matplotlib inline
```

executed in 446ms, finished 21:41:22 2019-10-24

In [2]:

```
from fsmc_code import compute_Phi_ET, simulate_hitting_time,
stationary_distribution
```

executed in 8ms, finished 21:41:22 2019-10-24

Exercise 2.1

What is the distribution of the number of fair coin tosses before one observes 3 heads in a row? To solve this, consider a 4-state Markov chain with transition probability matrix

$$P = \begin{bmatrix} 0.5 & 0.5 & 0 & 0 \\ 0.5 & 0 & 0.5 & 0 \\ 0.5 & 0 & 0 & 0.5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where $X_t = 1$ if the previous toss was tails, $X_t = 2$ if the last two tosses were tails then heads, $X_t = 3$ if the last three tosses were tails then heads twice, and $X_t = 4$ is an absorbing state that is reached when the last three tosses are heads.

- Write a computer program (e.g., in Python) to compute $\Pr(T_{1,4} = m)$ for $m = 1, 2, \ldots, 100$ and use this to estimate expected number of tosses $\mathbb{E}[T_{1,4}]$.

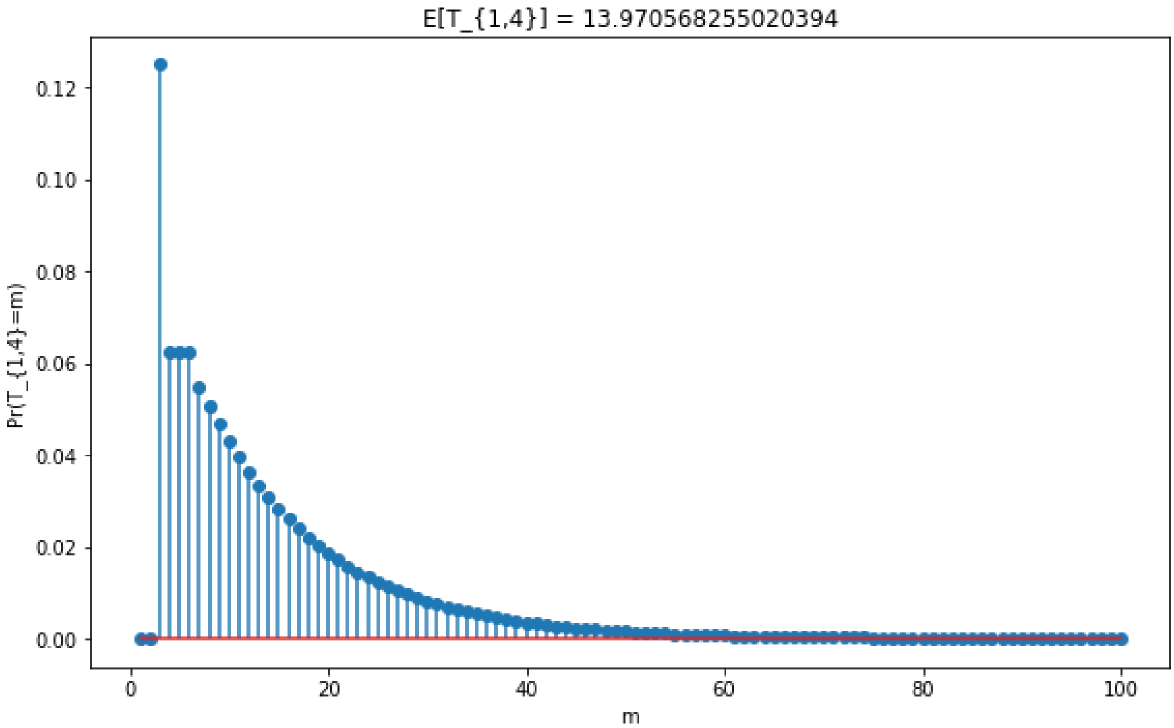
In [3]:

```
# See compute_Phi_ET in fsmc_code.py

P = np.array([[0.5, 0.5, 0, 0], [0.5, 0, 0.5, 0], [0.5, 0, 0, 0.5], [0, 0, 0,
1]])
Phi_list, ET = compute_Phi_ET(P, 100)

m = range(1,101)
Pr = [Phi_list[m][0][3]-Phi_list[m-1][0][3] for m in range(1, 101)]
E = ET[0][3]

plt.figure(figsize=(10, 6))
plt.stem(m, Pr, use_line_collection = True)
plt.xlabel(r'm')
plt.ylabel(r'Pr(T_{1,4}=m)')
plt.title(r'E[T_{1,4}] = ' + str(E))
```

executed in 242ms, finished 21:41:22 2019-10-24

Out[3]:

```
Text(0.5, 1.0, 'E[T_{1,4}] = 13.970568255020394')
```



- Write a computer program that generates 500 realizations from this Markov chain and uses them to plots a histogram of $T_{1,4}$.

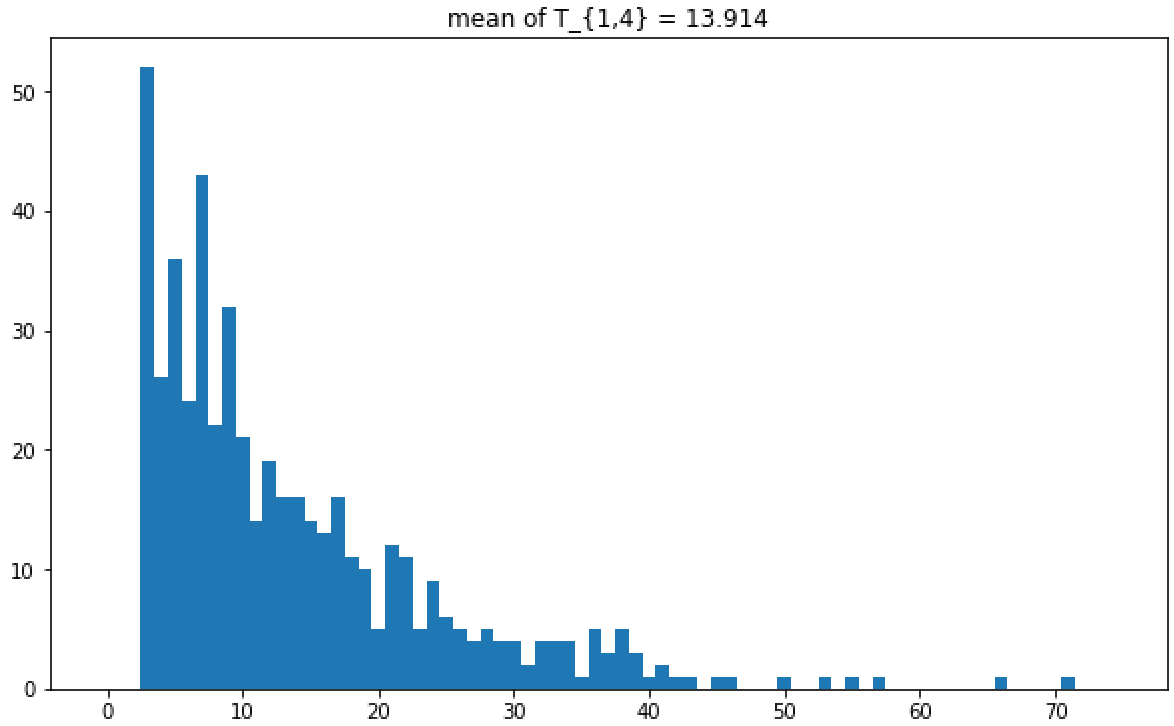
In [4]:

```
# implement simulate_hitting_time(P, states, nr) in fsmc_code.py

T = simulate_hitting_time(P, [0, 3], 500)
plt.figure(figsize=(10, 6))
plt.hist(T, bins=np.arange(max(T))-0.5)
plt.title(r'mean of T_{1,4} = ' + str(np.mean(T)))
```

executed in 288ms, finished 21:41:22 2019-10-24

Out[4]:

```
Text(0.5, 1.0, 'mean of T_{1,4} = 13.914')
```



Exercise 2.2

Consider the miniature chutes and ladders game shown in Figure 1. Assume a player starts on the space labeled 1 and plays by rolling a fair four-sided die and then moves that number of spaces. If a player lands on the bottom of a ladder, then they automatically climb to the top. If a player lands at the top of a slide, then they automatically slide to the bottom. This process can be modeled by a Markov chain with $n = 16$ states where each state is associated with a square where players can start their turn (e.g., players never start at the bottom of a ladder or the top of a slide). To finish the game, players must land exactly on space 20 (moves beyond this are not taken).

- Compute the transition probability matrix P of the implied Markov chain.

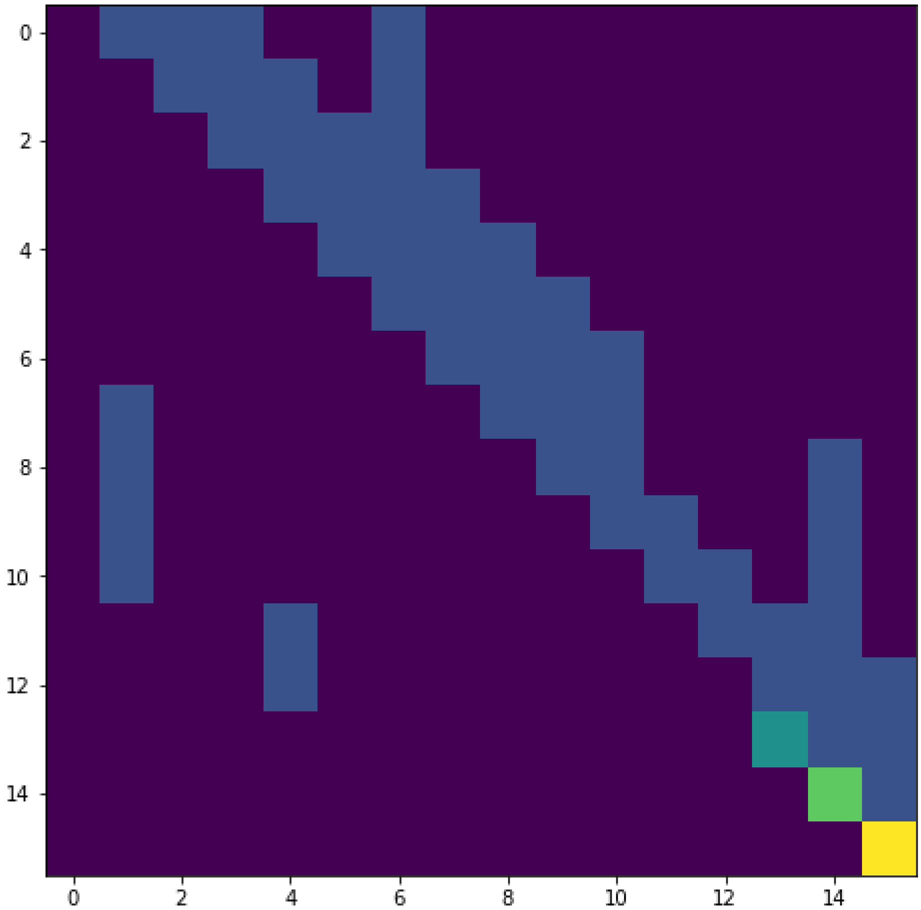
In [5]:

```
# You can either do this by hand (e.g., look at picture and write down matrix) or
by automating the process.

# By hand
P = np.asarray([[0,.25,.25,.25,0,0,.25,0,0,0,0,0,0,0,0,0],
                [0,0,.25,.25,.25,0,.25,0,0,0,0,0,0,0,0,0],
                [0,0,0,.25,.25,.25,.25,0,0,0,0,0,0,0,0,0],
                [0,0,0,0,.25,.25,.25,.25,0,0,0,0,0,0,0,0],
                [0,0,0,0,0,.25,.25,.25,.25,0,0,0,0,0,0,0],
                [0,0,0,0,0,0,.25,.25,.25,.25,0,0,0,0,0,0],
                [0,0,0,0,0,0,0,.25,.25,.25,.25,0,0,0,0,0],
                [0,.25,0,0,0,0,0,0,.25,.25,.25,0,0,0,0,0],
                [0,.25,0,0,0,0,0,0,0,.25,.25,0,0,0,.25,0],
                [0,.25,0,0,0,0,0,0,0,0,.25,.25,0,0,.25,0],
                [0,.25,0,0,0,0,0,0,0,0,0,.25,.25,0,.25,0],
                [0,0,0,0,.25,0,0,0,0,0,0,0,.25,.25,.25,0],
                [0,0,0,0,.25,0,0,0,0,0,0,0,0,.25,.25,.25],
                [0,0,0,0,0,0,0,0,0,0,0,0,0,.50,.25,.25],
                [0,0,0,0,0,0,0,0,0,0,0,0,0,0,.75,.25],
                [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1]])

###

# Or automated general function for Chutes and Ladders games
#def construct_P_matrix(n, dice, chutes, ladders):
'''
    Arguments:
        n {int} -- size of the state space
        dice {numpy.array} -- probability distribution of the dice outcome
        chutes {list[(int, int)]} -- the list of chutes, in pairs of (start, end)
        ladders {list[(int, int)]} -- the list of ladders, in pairs of (start,
end)

    Returns:
        P {numpy.array} -- n x n, transition matrix of the Markov chain
'''

    # Add code here to build matrix

 #   return P

n = 16  ### number of states
#dice = np.array([.25,.25,.25,.25])### probability distribution of dice
#chutes = [(13,2),(17,6)]### (sorce, destination) pairs of chutes
#ladders = [(4,8),(14,19)]### (sorce, destination) pairs of ladders
#P = construct_P_matrix(n, dice, chutes, ladders)


# Plot transition matrix
plt.figure(figsize=(8, 8))
plt.imshow(P)
```

executed in 231ms, finished 21:41:23 2019-10-24

Out[5]:

```
<matplotlib.image.AxesImage at 0x620317e10>
```



- For this Markov chain, write a computer program (e.g., in Python) to compute the cumulative distribution of the number turns a player takes to finish (i.e., the probability $\Pr(T_{1,20} \leq m)$ where $T_{1,20}$ is the hitting time from state 1 to state 20).

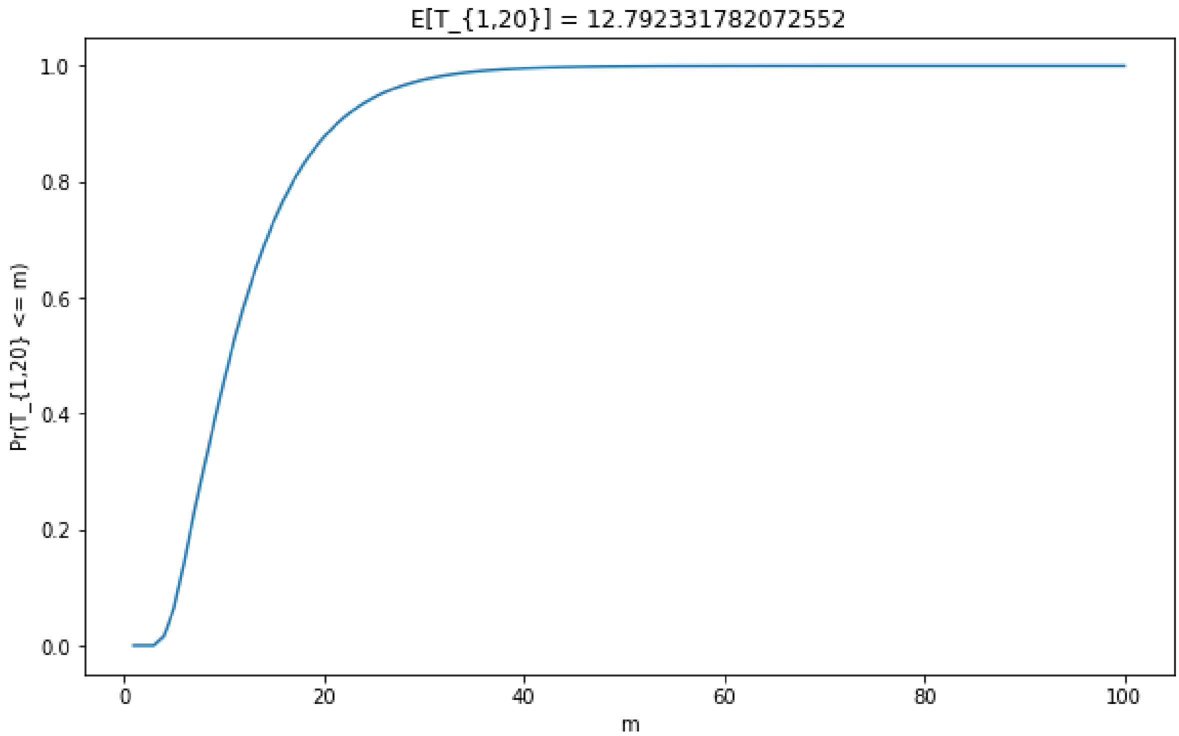
In [6]:

```
# Use previous functions to complete this exercise
Phi_list, ET = compute_Phi_ET(P, ns = 100)

m = range(1,101)### steps to be plotted
Pr = [Phi_list[m][0][n-1] for m in range(1,101)]  ### \Pr(T_{1,20} <= m) for all m
E = ET[0,15]### \mathbb{E}[T_{1,20}]

plt.figure(figsize=(10, 6))
plt.plot(m ,Pr)
plt.xlabel(r'm')
plt.ylabel(r'Pr(T_{1,20} <= m)')
plt.title(r'E[T_{1,20}] = ' + str(E))
```

executed in 661ms, finished 21:41:23 2019-10-24

Out[6]:

```
Text(0.5, 1.0, 'E[T_{1,20}] = 12.792331782072552')
```



- Write a computer program that generates 500 realizations from this Markov chain and uses them to plot a histogram of $T_{1,20}$.

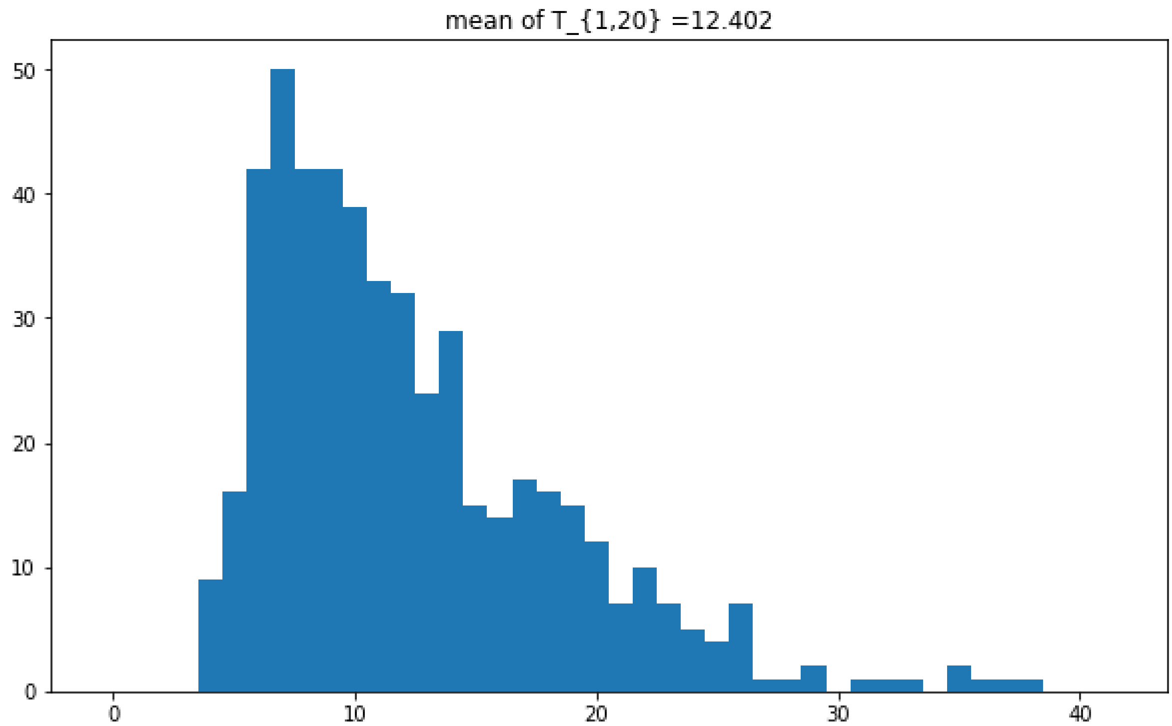
In [7]:

```
# Use previous funcitons to complete this exercise
T = simulate_hitting_time(P, [0, n-1], 500)
plt.figure(figsize=(10, 6))
plt.hist(T, bins=np.arange(max(T))-0.5)
plt.title(r'mean of T_{1,20} =' + str(np.mean(T)))
```

executed in 831ms, finished 21:41:24 2019-10-24

Out[7]:

```
Text(0.5, 1.0, 'mean of T_{1,20} =12.402')
```



- Optional Challenge: If the first player rolls 4 and climbs the ladder to square 8, then what is the probability that the second player will win.

In [8]:

```
# Use previous functions to complete this exercise

### compute Pr_win
```

executed in 2ms, finished 21:41:24 2019-10-24

Example 2.3

In a certain city, it is said that the weather is rainy with a 90% probability if it was rainy the previous day and with a 50% probability if it not rainy the previous day. If we assume that only the previous day's weather matters, then we can model the weather of this city by a Markov chain with $n = 2$ states whose transitions are governed by

$$P = \begin{bmatrix} 0.9 & 0.1 \\ 0.5 & 0.5 \end{bmatrix}$$

Under this model, what is the steady-state probability of rainy weather?

In [9]:

```
# implement stationary_distribution(P) in fsmc.py

P = np.array([[0.9, 0.1], [0.5, 0.5]])
stationary_distribution(P)
```

executed in 12ms, finished 21:41:24 2019-10-24

Out[9]:

```
array([[0.83333333],
       [0.16666667]])
```

Exercise 2.4

Consider a game where the gameboard has 8 different spaces arranged in a circle. During each turn, a player rolls two 4-sided dice and moves clockwise by a number of spaces equal to their sum. Define the transition matrix for this 8-state Markov chain and compute its stationary probability distribution.

In [10]:

```
# Use previous functions to complete this exercise
P = ([[1/16,0,1/16,1/8,3/16,1/4,3/16,1/8],[1/8,1/16,0,1/16,1/8,3/16,1/4,3/16],
[3/16,1/8,1/16,0,1/16,1/8,3/16,1/4],[1/4,3/16,1/8,1/16,0,1/16,1/8,3/16],
[3/16,1/4,3/16,1/8,1/16,0,1/16,1/8],[1/8,3/16,1/4,3/16,1/8,1/16,0,1/16],
[1/16,1/8,3/16,1/4,3/16,1/8,1/16,0],[0,1/16,1/8,3/16,1/4,3/16,1/8,1/16]])###
construct the transition matrix
print(P)
stationary_distribution(P)
```

executed in 17ms, finished 21:41:24 2019-10-24

```
[[0.0625, 0, 0.0625, 0.125, 0.1875, 0.25, 0.1875, 0.125], [0.125, 0.06
25, 0, 0.0625, 0.125, 0.1875, 0.25, 0.1875], [0.1875, 0.125, 0.0625,
0, 0.0625, 0.125, 0.1875, 0.25], [0.25, 0.1875, 0.125, 0.0625, 0, 0.06
25, 0.125, 0.1875], [0.1875, 0.25, 0.1875, 0.125, 0.0625, 0, 0.0625,
0.125], [0.125, 0.1875, 0.25, 0.1875, 0.125, 0.0625, 0, 0.0625], [0.06
25, 0.125, 0.1875, 0.25, 0.1875, 0.125, 0.0625, 0], [0, 0.0625, 0.125,
0.1875, 0.25, 0.1875, 0.125, 0.0625]]
```

Out[10]:

```
array([[0.125],
       [0.125],
       [0.125],
       [0.125],
       [0.125],
       [0.125],
       [0.125],
       [0.125]])
```

Next, suppose that one space is special (e.g., state-1 of the Markov chain) and a player can only leave this space by rolling doubles (i.e., when both dice show the same value). Again, the player moves clockwise by a number of spaces equal to their sum. Define the transition matrix for this 8-state Markov chain and compute its stationary probability distribution.

In [11]:

```
# Use previous functions to complete this exercise
P = ([[13/16,0,1/16,0,1/16,0,1/16,0],[1/8,1/16,0,1/16,1/8,3/16,1/4,3/16],
[3/16,1/8,1/16,0,1/16,1/8,3/16,1/4],[1/4,3/16,1/8,1/16,0,1/16,1/8,3/16],
[3/16,1/4,3/16,1/8,1/16,0,1/16,1/8],[1/8,3/16,1/4,3/16,1/8,1/16,0,1/16],
[1/16,1/8,3/16,1/4,3/16,1/8,1/16,0],[0,1/16,1/8,3/16,1/4,3/16,1/8,1/16]])###
construct the transition matrix
print(P)
stationary_distribution(P)
```

executed in 15ms, finished 21:41:24 2019-10-24

```
[[0.8125, 0, 0.0625, 0, 0.0625, 0, 0.0625, 0], [0.125, 0.0625, 0, 0.06
25, 0.125, 0.1875, 0.25, 0.1875], [0.1875, 0.125, 0.0625, 0, 0.0625,
0.125, 0.1875, 0.25], [0.25, 0.1875, 0.125, 0.0625, 0, 0.0625, 0.125,
0.1875], [0.1875, 0.25, 0.1875, 0.125, 0.0625, 0, 0.0625, 0.125], [0.1
25, 0.1875, 0.25, 0.1875, 0.125, 0.0625, 0, 0.0625], [0.0625, 0.125,
0.1875, 0.25, 0.1875, 0.125, 0.0625, 0], [0, 0.0625, 0.125, 0.1875, 0.
25, 0.1875, 0.125, 0.0625]]
```

Out[11]:

```
array([[0.41836864],
       [0.08285234],
       [0.10176963],
       [0.07092795],
       [0.09311176],
       [0.0625555 ],
       [0.09593429],
       [0.07447989]])
```