MP307 Practical 1 Queueing Theory I

In [ ]:

In [2]:
```python
# import a standard Python package for doing matrix theory below
import numpy as np
```

In [ ]:

Q.1

Consider the two state telephone example considered in lecture 2 with transition matrix

$$P = \begin{bmatrix} 1/2 & 1/2 \\ 2/3 & 1/3 \end{bmatrix}$$

This can be stored in a Python array `P` as follows:

In [3]:
```python
P=np.array([[1/2,1/2],[2/3,1/3]])
print(P)
```

```
[[0.5        0.5       ]
 [0.66666667 0.33333333]]
```

$\mathrm{Prob}(i \to j)$ in 2 time steps is found by evaluating P^2.
We may multiply two matrices using the `np.matmul` Python function.

In [4]:
```python
np.matmul(P,P)
```

Out[4]:
```
array([[0.58333333, 0.41666667],
       [0.55555556, 0.44444444]])
```

We may also take the power of a matrix using the `np.linalg.matrix_power` Python function

In [5]:
```python
np.linalg.matrix_power(P,2)
```

Out[5]:
```
array([[0.58333333, 0.41666667],
       [0.55555556, 0.44444444]])
```

In [6]:
```python
np.linalg.matrix_power(P,5)
```

Out[6]:
```
array([[0.57137346, 0.42862654],
       [0.57150206, 0.42849794]])
```

Notice that the rows are converging which suggests that the system is ergodic i.e. for $p_m(i,j)$ the probability of transtition from state i to state j in m steps then

$$\lim_{m\to\infty} p_m(i,j) = \pi_j$$

exists for equilibrium probability π_j independent of the initial state i.

In fact the above system is ergodic with equilibrium probabilities $\pi_0 = 4/7$ and $\pi_1 = 3/7$

In [7]:
```
print(4/7)
print(3/7)
```

```
0.5714285714285714
0.42857142857142855
```

The equilibrium probabilities are found by looking for the **left** eigenvector of P for eigenvalue 1.
We can find all eigenvectors and eigenvalues of P by use of the `np.linalg.eig` Python function as follows.

In [8]:
```
eigendata=np.linalg.eig(P)
print(eigendata)
```

```
(array([ 1.        , -0.16666667]), array([[ 0.70710678, -0.6       ],
       [ 0.70710678,  0.8       ]]))
```

The output is a tuple consisting of an array and a matrix. We may unpack the tuple as follows:

In [9]:
```
X, V=eigendata
```

In [10]:
```
print(X)
print(V)
```

```
[ 1.         -0.16666667]
[[ 0.70710678 -0.6       ]
 [ 0.70710678  0.8       ]]
```

The eigenvalues of P are given in the array `X` with eigenvectors given in the matrix `V`. In particular the eigenvector for 1 is the first column vector with equal entries. We can read this off the entries of the first column as an array:

In [11]:
```
u=V[:,0] # Note that Python indices run as 0,1, ...
print(u)
```

```
[0.70710678 0.70710678]
```

A constant scalar multiple of an eigenvector is also an eigenvector so in fact we have confirmed that $Pu = u$ for $u = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$.

Thus the `np.linalg.eig` Python function returns the **right** eigenvectors of P. Therefore we consider the right eigenvectors of the transpose of P, denoted by P^T, which provide us with the left eigenvectors of P.

In [12]:
```
PT=P.transpose()
print(PT)
```

```
[[0.5        0.66666667]
 [0.5        0.33333333]]
```

In [13]:
```
eigendata=np.linalg.eig(PT)
```

In [14]:
```
X, V=eigendata
print(X)
print(V)
```

```
[ 1.         -0.16666667]
[[ 0.8        -0.70710678]
 [ 0.6         0.70710678]]
```

Thus the left eigenvector of P for eigenvalue 1 is

In [15]:
```
u=V[:,0]
print(u)
print(u.real) # print real part of u
```

```
[0.8 0.6]
[0.8 0.6]
```

We normalize this eigenvector so that the entries sum to unity. We thus obtain the equilibrium probability vector π

In [16]:
```
pi=u/sum(u) # divide elements of u by sum of elements of u
print(pi.real)
print([4/7,3/7])
```

```
[0.57142857 0.42857143]
[0.5714285714285714, 0.42857142857142855]
```

In [ ]:
```
```

Q.2 (*)

A finite queue of maximum size 3 is observed with the following transition matrix

$$\begin{bmatrix} 1/3 & 0 & 2/5 & 4/15 \\ 1/4 & 0 & 3/10 & 9/20 \\ 0 & 2/3 & 1/5 & 2/15 \\ 1/5 & 0 & 2/5 & 2/5 \end{bmatrix}$$

1. Find Prob($i \to j$ in 10 steps).

2. Find the equilibrium probabilities $\pi_0, \pi_1, \pi_2, \pi_3$.

In [17]:
```
P=np.array([
[1/3,0,2/5,4/15],
[1/4,0,3/10,9/20],
[0,2/3,1/5,2/15],
[1/5,0,2/5,2/5]
])
print(P)
```

```
[[0.33333333 0.         0.4        0.26666667]
 [0.25       0.         0.3        0.45      ]
 [0.         0.66666667 0.2        0.13333333]
 [0.2        0.         0.4        0.4       ]]
```

In [18]:
```
np.linalg.matrix_power(P,10)
```

Out[18]:
```
array([[0.17004087, 0.21052518, 0.31578941, 0.30364454],
       [0.17004046, 0.21052647, 0.315789  , 0.30364406],
       [0.17003993, 0.21052791, 0.31578988, 0.30364227],
       [0.17004086, 0.21052518, 0.31578941, 0.30364455]])
```

Notice that the rows are converging which suggests that the system is ergodic

Repeat the sequence of steps followed in Q.1 above.

Now we find the equilibrium probabilities. They are found by looking for the **left** eigenvector of P for eigenvalue 1

In [19]:
```
eigendata=np.linalg.eig(P)
print(eigendata)
```

```
(array([ 1.        +0.j        , -0.1       +0.23804761j,
       -0.1       -0.23804761j,  0.13333333+0.j        ]), array([[ 0.5       +0.j
, -0.37558431-0.12772421j,
        -0.37558431+0.12772421j, -0.91195147+0.j        ],
       [ 0.5       +0.j        , -0.25486078+0.28737948j,
        -0.25486078-0.28737948j, -0.08797872+0.j        ],
       [ 0.5       +0.j        ,  0.73328365+0.j        ,
         0.73328365-0.j        ,  0.24407   +0.j        ],
       [ 0.5       +0.j        , -0.37558431-0.12772421j,
        -0.37558431+0.12772421j,  0.3178586 +0.j        ]]))
```

The output is a tuple consisting of an array and a matrix. We may unpack the tuple as follows:

In [20]:
```
X, V=eigendata
```

In [21]:
```
print(X)
print(V)
```

```
[ 1.        +0.j         -0.1       +0.23804761j -0.1       -0.23804761j
  0.13333333+0.j        ]
[[ 0.5       +0.j         -0.37558431-0.12772421j -0.37558431+0.12772421j
  -0.91195147+0.j        ]
 [ 0.5       +0.j         -0.25486078+0.28737948j -0.25486078-0.28737948j
  -0.08797872+0.j        ]
 [ 0.5       +0.j          0.73328365+0.j          0.73328365-0.j
   0.24407   +0.j        ]
 [ 0.5       +0.j         -0.37558431-0.12772421j -0.37558431+0.12772421j
   0.3178586 +0.j        ]]
```

The eigenvalues of P are given in the array X with eigenvectors given in the matrix V. In particular the eigenvector for 1 is the first column vector with equal entries. We can read this off the entries of the first column as an array:

In [22]:
```
u=V[:,0] # Note that Python indices run as 0,1, ...
print(u)
```

```
[0.5+0.j 0.5+0.j 0.5+0.j 0.5+0.j]
```

A constant scalar multiple of an eigenvector is also an eigenvector so in fact we have confirmed that $Pu = u$ for $u = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$.

Thus the `np.linalg.eig` Python function returns the **right** eigenvectors of P. Therefore we consider the right eigenvectors of the transpose of P, denoted by P^T, which provide us with the left eigenvectors of P.

In [23]:
```
PT=P.transpose()
print(PT)
```

```
[[0.33333333 0.25       0.         0.2        ]
 [0.         0.         0.66666667 0.         ]
 [0.4        0.3        0.2        0.4        ]
 [0.26666667 0.45       0.13333333 0.4        ]]
```

In [26]:
```
eigendata=np.linalg.eig(PT)
```

In [27]:
```
X, V=eigendata
print(X)
print(V)
```

```
[ 1.        +0.j          -0.1       +0.23804761j -0.1       -0.23804761j
  0.13333333+0.j        ]
[[ 3.30217662e-01+0.j          -2.50201620e-01-0.01669904j
  -2.50201620e-01+0.01669904j  7.07106781e-01+0.j        ]
 [ 4.08840914e-01+0.j           7.79444298e-01+0.j
   7.79444298e-01-0.j          -1.31612011e-16+0.j        ]
 [ 6.13261372e-01+0.j          -1.16916645e-01+0.27831728j
  -1.16916645e-01-0.27831728j -1.24265400e-16+0.j        ]
 [ 5.89674396e-01+0.j          -4.12326034e-01-0.26161825j
  -4.12326034e-01+0.26161825j -7.07106781e-01+0.j        ]]
```

Thus the left eigenvector of P for eigenvalue 1 is

In [28]:
```
u=V[:,0]
print(u)
print(u.real) # print real part of u
```

```
[0.33021766+0.j 0.40884091+0.j 0.61326137+0.j 0.5896744 +0.j]
[0.33021766 0.40884091 0.61326137 0.5896744 ]
```

We normalize this eigenvector so that the entries sum to unity. We thus obtain the equilibrium probability vector π

In [29]:
```
pi=u/sum(u) # divide elements of u by sum of elements of u
print(pi.real)
```

```
[0.17004049 0.21052632 0.31578947 0.30364372]
```

In [ ]:

Q.3 (*)

Consider the random walk on 6 sites with the following transition matrix

$$\begin{bmatrix} 1/2 & 0 & 1/2 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 1/2 & 0 & 0 \\ 0 & 0 & 1/2 & 0 & 1/2 & 0 \\ 0 & 1/4 & 0 & 1/2 & 0 & 1/4 \\ 1/4 & 0 & 1/4 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 1/2 & 0 & 1/2 \end{bmatrix}$$

(a) Is the system ergodic?

(b) Compare your result to that for the modified random walk with transition matrix below.

$$\begin{bmatrix} 1/4 & 1/4 & 1/2 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 1/2 & 0 & 0 \\ 0 & 0 & 1/2 & 0 & 1/2 & 0 \\ 0 & 1/4 & 0 & 1/2 & 0 & 1/4 \\ 1/4 & 0 & 1/4 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 1/2 & 0 & 1/2 \end{bmatrix}$$

Explain the observed difference in behaviour.

In [48]:
```
P=np.array([
[1/2, 0, 1/2, 0, 0, 0],
[0,1/2, 0, 1/2, 0, 0],
[0, 0, 1/2, 0, 1/2, 0],
[0,1/4, 0, 1/2, 0, 1/4],
[1/4, 0, 1/4, 0, 1/2, 0],
[0, 0, 0, 1/2, 0, 1/2]
])
print(P)
```

```
[[0.5  0.   0.5  0.   0.   0.  ]
 [0.   0.5  0.   0.5  0.   0.  ]
 [0.   0.   0.5  0.   0.5  0.  ]
 [0.   0.25 0.   0.5  0.   0.25]
 [0.25 0.   0.25 0.   0.5  0.  ]
 [0.   0.   0.   0.5  0.   0.5 ]]
```

(a) is the system ergodic?

In [49]:
```
np.linalg.matrix_power(P,100)
```

Out[49]:
```
array([[0.2 , 0.  , 0.4 , 0.  , 0.4 , 0.  ],
       [0.  , 0.25, 0.  , 0.5 , 0.  , 0.25],
       [0.2 , 0.  , 0.4 , 0.  , 0.4 , 0.  ],
       [0.  , 0.25, 0.  , 0.5 , 0.  , 0.25],
       [0.2 , 0.  , 0.4 , 0.  , 0.4 , 0.  ],
       [0.  , 0.25, 0.  , 0.5 , 0.  , 0.25]])
```

The rows are not converging which suggests that the system is not ergodic

(b) Compare your result to that for the modified random walk with transition matrix below

In [46]:
```
P1=np.array([
[1/4, 1/4, 1/2, 0, 0, 0],
[0,1/2, 0, 1/2, 0, 0],
[0, 0, 1/2, 0, 1/2, 0],
[0,1/4, 0, 1/2, 0, 1/4],
[1/4, 0, 1/4, 0, 1/2, 0],
[0, 0, 0, 1/2, 0, 1/2]
])
print(P1)
```

```
[[0.25 0.25 0.5  0.   0.   0.  ]
 [0.   0.5  0.   0.5  0.   0.  ]
 [0.   0.   0.5  0.   0.5  0.  ]
 [0.   0.25 0.   0.5  0.   0.25]
 [0.25 0.   0.25 0.   0.5  0.  ]
 [0.   0.   0.   0.5  0.   0.5 ]]
```

In [47]:
```
np.linalg.matrix_power(P1,100)
```

Out[47]:
```
array([[0.00229594, 0.24705701, 0.00602824, 0.49227287, 0.00653303,
         0.24581291],
        [0.        , 0.25      , 0.        , 0.5       , 0.        ,
         0.25      ],
        [0.00326651, 0.24581291, 0.00857658, 0.48900635, 0.00929476,
         0.24404289],
        [0.        , 0.25      , 0.        , 0.5       , 0.        ,
         0.25      ],
        [0.00301412, 0.24613643, 0.00791389, 0.4898558 , 0.00857658,
         0.24450318],
        [0.        , 0.25      , 0.        , 0.5       , 0.        ,
         0.25      ]])
```

In [ ]:

Q.4 (*)

A queue is observed over 1000 time intervals where the size of the queue after each time step is given. Construct a simple model for this queue as a Markov chain with only nearest neighbour interactions.

1. What is the expected behaviour of the queue as time continues?

2. Is the system ergodic?

In [76]:
```
qdata = [4, 5, 6, 6, 6, 7, 6, 7, 6, 5, 4, 4, 5, 6, 7, 6, 5, 4, 3, 4, 5, 6, 5, 4, 3,
          2, 3, 4, 3, 2, 3, 2, 1, 1, 2, 2, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 2, 3,
          8, 7, 6, 7, 6, 6, 5, 4, 5, 4, 3, 2, 3, 2, 3, 2, 3, 2, 1, 1, 2, 3, 3, 4, 5,
          6, 5, 6, 5, 4, 5, 4, 3, 4, 3, 4, 3, 2, 1, 0, 0, 0, 0, 1, 2, 1, 0, 1, 2, 3,
          3, 2, 1, 2, 3, 2, 3, 2, 1, 0, 0, 1, 1, 2, 3, 2, 1, 0, 0, 1, 0, 0, 0, 0, 1,
          1, 2, 3, 4, 3, 4, 5, 6, 5, 6, 5, 6, 7, 7, 6, 5, 4, 3, 4, 3, 3, 4, 3, 2, 3,
          2, 1, 0, 0, 0, 0, 1, 2, 3, 2, 2, 3, 2, 1, 0, 1, 2, 1, 0, 1, 0, 1, 0, 1, 2,
          0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 2, 1, 1, 1, 0, 0, 1, 2, 3, 2, 3, 3, 4, 5, 6,
          1, 0, 0, 0, 1, 2, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1,
          1, 0, 0, 1, 2, 3, 2, 3, 2, 2, 3, 4, 4, 5, 4, 3, 2, 3, 2, 3, 2, 3, 2, 1, 0,
          1, 0, 1, 0, 1, 2, 3, 3, 2, 1, 1, 2, 1, 2, 1, 2, 1, 2, 3, 2, 2, 1, 0, 1, 2,
          2, 2, 1, 2, 3, 4, 5, 4, 4, 5, 4, 3, 4, 3, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          1, 2, 2, 3, 2, 3, 4, 5, 6, 5, 6, 5, 4, 5, 6, 7, 6, 5, 6, 6, 7, 6, 7, 7, 6,
          4, 5, 6, 6, 5, 6, 5, 6, 5, 6, 5, 5, 5, 4, 3, 4, 5, 6, 5, 4, 3, 3, 4, 3, 2,
          2, 1, 0, 0, 0, 0, 0, 1, 2, 3, 2, 3, 4, 5, 6, 7, 7, 8, 9, 8, 9, 8, 7, 6, 5,
          1, 0, 1, 2, 1, 2, 3, 4, 5, 6, 7, 7, 8, 7, 8, 9, 8, 7, 6, 5, 4, 3, 4, 3, 4,
          2, 3, 2, 3, 3, 2, 1, 1, 0, 1, 2, 1, 2, 2, 3, 4, 5, 6, 5, 4, 3, 2, 1, 0, 1,
          3, 2, 1, 0, 0, 1, 2, 3, 4, 3, 2, 3, 2, 2, 1, 0, 1, 0, 0, 1, 0, 1, 2, 3, 4,
          6, 7, 8, 8, 9, 10, 9, 10, 9, 10, 11, 10, 9, 8, 9, 10, 11, 10, 10, 9, 10, 1
          9, 8, 7, 6, 7, 6, 7, 6, 5, 4, 4, 5, 4, 3, 2, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1,
          0, 0, 1, 0, 0, 1, 0, 1, 2, 3, 2, 3, 2, 3, 4, 3, 2, 1, 1, 2, 1, 2, 1, 2, 1,
          2, 2, 1, 0, 1, 2, 3, 4, 5, 4, 5, 4, 3, 2, 3, 4, 5, 5, 4, 3, 4, 3, 4, 5, 4,
          3, 4, 3, 2, 3, 4, 3, 2, 3, 2, 1, 0, 0, 0, 0, 0, 1, 0, 1, 2, 3, 4, 5, 4, 5,
          6, 5, 4, 3, 2, 1, 0, 0, 1, 0, 0, 1, 0, 1, 2, 3, 4, 3, 4, 3, 2, 1, 1, 2, 3,
          4, 3, 4, 4, 5, 6, 7, 6, 7, 7, 6, 5, 4, 3, 4, 3, 2, 2, 3, 4, 3, 4, 3, 2, 1,
          1, 2, 3, 2, 1, 0, 0, 0, 0, 1, 1, 2, 3, 2, 3, 2, 3, 4, 5, 6, 5, 5, 4, 3, 4,
          3, 2, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 2, 1, 0, 0, 0, 1, 0, 0, 1, 2, 3,
          3, 2, 3, 2, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 2, 1, 2, 1, 0, 1, 2, 1, 2, 1, 2,
          0, 0, 1, 2, 2, 3, 3, 2, 3, 4, 3, 2, 1, 2, 3, 3, 2, 1, 0, 1, 2, 2, 3, 4, 5,
          5, 4, 5, 4, 3, 2, 3, 4, 3, 2, 1, 2, 3, 4, 3, 2, 3, 2, 1, 0, 1, 2, 1, 0, 0,
          0, 1, 2, 2, 3, 2, 1, 0, 1, 2, 3, 4, 4, 3, 4, 3, 2, 1, 0, 0, 0, 0, 1, 0, 1,
          0, 0, 1, 0, 1, 2, 2, 3, 3, 4, 3, 2, 3, 2, 1, 0, 0, 1, 2, 1, 2, 3, 2, 1, 2,
          1, 2, 1, 2, 3, 2, 3, 2, 1, 1, 0]
```

In [77]:
```
N=len(qdata)
```

In [78]:
```
N
```

Out[78]:
```
1000
```

N is the number of events recorded

We can easily compute the average with the following do loop

In [79]:
```
ans=0
for i in range(1,N):
    ans=ans+qdata[i-1]
#do loop complete
average=ans/N
```

In [80]:
```
average
```

Out[80]:
```
2.718
```

In [ ]:
```

```

Apply the simplest model of an infinite nearest neighbour queue with p_1 = prob of jumping up and p_2 = prob of jumping down. These can be estimated by counting the number of jumps up vs the number of jumps down. Consider the following do loop with if statements.

In [44]:
```
nup=0 # nup counts no of up steps
ndown=0 # ndown counts no of down steps

for i in range(1,N):
    if qdata[i]>qdata[i-1]:
        nup=nup+1
    elif qdata[i]<qdata[i-1]:
        ndown=ndown+1
# if statements complete
print(nup,"up steps", ndown,"down steps")
```

```
417 up steps 421 down steps
```

In [45]:
```
p1=nup/N
p2=ndown/N
print(p1,p2)
```

```
0.417 0.421
```

Do you consider this to be a good estimate? What is wrong?

There is no consideration for the queue to stay the same, i.e. the 1-p1-p2 scenario. Modified code below

In [81]:
```
nup=0 # nup counts no of up steps
ndown=0 # ndown counts no of down steps
not0 = 0 # counts no of nonzero states


for i in range(1,N):
     if qdata[i] > 0:
        not0 = not0 + 1
        if qdata[i]>qdata[i-1]:
```

```
            nup=nup+1
        elif qdata[i]<qdata[i-1]:
            ndown=ndown+1
# if statements complete
print(nup,"up steps", ndown,"down steps")
```

```
417 up steps 337 down steps
```

In [82]:
```
p1 = nup / not0
p2 = ndown / not0
print(p1 / p2)
```

```
1.2373887240356085
```

In [83]:
```
p1
```

Out[83]:
```
0.5036231884057971
```

In [84]:
```
p2
```

Out[84]:
```
0.40700483091787437
```

Q.5 (*)

A queue is observed over 10000 one-second time intervals with data as given below in the array `qdata`.

Construct a Poisson nearest neighbour model with a single arrival and servicing pattern and hence answer the following questions:

1. What is the average time taken for 1 customer to arrive?
2. What is the average number of customer servicings per second?
3. What is your estimate for the equilibrium probability $P(n \geq 4)$, where n is the queue size in this model?
4. Suppose that two equivalent servers are introduced. What would the equilibrium probability $P(n \geq 4)$ then be?

In [69]:
```
qdata = [5, 5, 5, 6, 5, 6, 6, 7, 6, 5, 6, 5, 4, 5, 6, 6, 6, 7, 8, 7, 6, 6, 6, 5, 4,
```

In [70]:
```
m = len(qdata)
```

In [71]:
```
nup = 0 # nup counts no of up steps
ndown = 0 #ndown counts no of odwn steps
not0 = 0 # counts no of nonzero states

for i in range (1,m):
    if qdata[i] > 0:
        not0 = not0 + 1
        if qdata[i] > qdata[i-1]:
            nup = nup + 1
        elif qdata[i] < qdata[i-1]:
            ndown = ndown + 1
# no of if statements complete
print(not0, 'non zero states', nup, 'up steps', ndown, 'downsteps')
```

```
7822 non zero states 3068 up steps 2408 downsteps
```

In [73]:
```
p1 = nup / not0
p2 = ndown / not0
print(p1,p2)
print(p1 / p2)
```

```
0.39222705190488366 0.3078496548197392
1.2740863787375416
```

What is the average time taken for 1 customer to arrive?

In [62]:
```
ans=0
for i in range(1,N):
    ans=ans+qdata[i-1]
#do loop complete
average=ans/N
```

In [63]:
```
average
```

Out[63]:
```
3.783
```

What is the average number of customer servicings per second?

In [64]:
```

```

In [74]:
```
p = (p1 / p2)**4
print(p)
```

```
2.635090229848321
```

In [75]:
```
#4
w = (p1 / (2 * p2))**4
print(w)
```

```
0.16469313936552007
```

In [ ]:
```

```