Costco Time Markov Chains

Prompt

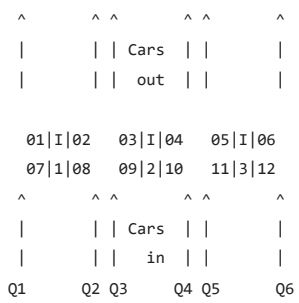
Original Prompt can be found here. A copy of the prompt along with the completed exercise can be found under /Applications.

Summary

Cars arrive at the Costco gas station at rate $\lambda = 1$ car per minute.

There are 3 islands with 2 pumps on each side, for a total of 12 pumps. Label the pumps in the back from left to right as 1, 2, ..., 6, and the pumps in front from left to right as 7, 8, ..., 12.

When cars arrive they join one of 6 queues, one queue for each side of the islands. There is one queue that waits for pumps 1 and 7, one that waits for pumps 2 and 8, and so on. The car first in line in the queue for pumps 1 and 7 will move to whichever one opens up first, similarly for 2 and 8, and so on.



There are 3 different types of cars: 30% of cars will only join the queues on the left (to wait for one of the odd numbered pumps), 30% of cars will only join the queues on the right (to wait for an even numbered pump), and the remaining cars will join any queue.

When a car arrives, it will join whichever queue of its type is the shortest. If there is a tie for shortest queue of its type, the car will choose at random among the shortest queue options. Once a car chooses a queue, it does not switch to another or leave the gas station before completing service.

Assume each pump serves customers at Exponential rate $\mu = 0.2$ cars per minute, independent of the customer type. You can assume that once a car completes service the next car in the queue starts service immediately.

Part 1

Let $X_i(t)$ denote the number of customers in queue $i = 1, \ldots, 6$ at time t, including any customers in service at the corresponding pumps. The process $X(t) = (X_1(t), \ldots, X_6(t))$ is a vector-valued continuous time Markov chain. Let $S(t) = X_1(t) + \cdots + X_6(t)$ denote the total number of customers in the system at time t, including any customers in service.

Note: t is in minutes

Write a program to simulate the customer arrivals and services and the values of $X_i(t)$ over a long time period, say a week. You will use your simulation results to approximate the following items, so consider all parts below when designing your code.

- The long run distribution of the number of cars in the system.
- The long run fraction of time there are no cars in the system.
- The long run average number of customers in the system.
- Any other features of the long run distribution of the number of cars in the system you're interested in.
- The long run distribution of the amount of time a customer spends in the system.

- The average time a customer spends in the system.
- Any other features of the long run distribution of the amount of time a customer spends in the system you're interested in

Assignment Update: The last three bullet points are optional.

Part 2

Change some features of the set up and investigate how your changes effect the average number of customers in the system, and the average time a customer spends in the system. There are lots of things you can do.

Application

Part 1

```
import numpy as np

# Variables to describe the system
pumps_total = 12
pumps_queue = 2
queues = 6
rate_arrival_total = 1
rate_service_pump = 0.2
rate_service_total = 0.2 * pumps_total

ra_lambda = rate_arrival_total

rs_mu = rate_service_total

#Rates are per minute and we are simulating for a week
min_in_week = 60*24*7

n_jumps = 30000

X_t = np.zeros([n_jumps, queues])
W_n = np.zeros(n_jumps)
T_n = np.zeros(n_jumps)

for n in range(1,n_jumps):
    #generate value from exp dist
    Y = np.random.exponential(scale=1)

    #determine cars at pumps
    serving_cars = 0
    for q in range(0,queues):
      serving_cars = serving_cars +  min(pumps_queue, X_t[n - 1,q])

    #determine waiting time and total elapsed time
    roos_total = ra_lambda + rate_service_pump * serving_cars # roos = rate out of state
    W_n[n - 1] = Y / roos_total
    T_n[n] = T_n[n - 1] + W_n[n - 1]

    #determine if event is a service or arrival
    #determine probabilities
    p_arrival = ra_lambda/roos_total
    p_service_1 = rate_service_pump*min(pumps_queue, X_t[n - 1,1-1])/roos_total
    p_service_2 = rate_service_pump*min(pumps_queue, X_t[n - 1,2-1])/roos_total
    p_service_3 = rate_service_pump*min(pumps_queue, X_t[n - 1,3-1])/roos_total
    p_service_4 = rate_service_pump*min(pumps_queue, X_t[n - 1,4-1])/roos_total
    p_service_5 = rate_service_pump*min(pumps_queue, X_t[n - 1,5-1])/roos_total
    p_service_6 = rate_service_pump*min(pumps_queue, X_t[n - 1,6-1])/roos_total
    p_event = [p_service_1, p_service_2, p_service_3, p_service_4, p_service_5, p_service_6, p_arrival]

    p_total = sum(p_event)
    # if p_total != 1:
    #   print("prob total error")
    #   print(p_total)

    #generate random event
    event = np.random.choice([0, 1, 2, 3, 4, 5, 6], replace=True, p=p_event)

    #check if arrival or service and adjust X accordingly
    if event == 6:
      #print("arrival")
      #arrival - determine if orientation preference and decide on queue
      side = np.random.choice(["left", "right", "either"], replace=True, p=[0.3, 0.3, 0.4])

      line = np.zeros(0)
      queue_selection = -1

      if side == "left":
        #print("left")
        left_queues = [X_t[n - 1,1-1], X_t[n - 1,3-1], X_t[n - 1,5-1]]
        left_min = min(left_queues)

        for i in range(0,3):
          if left_queues[i]==left_min:
            line = np.append(line,i)


        queue_selection = int(np.random.choice(line))*2
```

```
      elif side == "right":
        #print("right")
        right_queues = [X_t[n - 1,2-1], X_t[n - 1,4-1], X_t[n - 1,6-1]]
        right_min = min(right_queues)

        for i in range(0,3):
          if right_queues[i]==right_min:
            line = np.append(line,i)


        queue_selection = int(np.random.choice(line))*2 + 1

      else:
        #print("either")
        all_queues = [X_t[n - 1,1-1], X_t[n - 1,2-1], X_t[n - 1,3-1], X_t[n - 1,4-1], X_t[n - 1,5-1], X_t[n - 1,6-1]]
        all_min = min(all_queues)

        for i in range(0,6):
          if all_queues[i]==all_min:
            line = np.append(line,i)

        queue_selection = int(np.random.choice(line))

      X_t[n,:] = X_t[n - 1,:]
      X_t[n,queue_selection] = X_t[n - 1,queue_selection] + 1

    else:
      #print("service")
      #service - take customer out of queue
      X_t[n,:] = X_t[n - 1,:]
      X_t[n,event] = X_t[n - 1,event] - 1

    #print(X_t[n,:])


# Find the index where T_n exceeds min_in_week
first_index_exceeding_min = np.argmax(T_n > min_in_week)

# print(T_n[first_index_exceeding_min-1] > min_in_week)
# print(T_n[first_index_exceeding_min] > min_in_week)
# print(T_n[first_index_exceeding_min+1] > min_in_week)

print("A week is reached in T_n at index:", first_index_exceeding_min)
```

```
A week is reached in T_n at index: 19845
```

The long run distribution of the number of cars in the system.

```
# cut data to longer than week time span (should be big though)
import pandas as pd

if (first_index_exceeding_min != 0):
  data = X_t[0:first_index_exceeding_min,:]
  jumps = first_index_exceeding_min
else:
  data = X_t
  jumps = n_jumps


row_sums = np.sum(data, axis=1)
max_customers = int(max(row_sums))
time_with_x_cars_in_sys = np.zeros(max_customers+1)

for j in range(0,jumps):
  row = data[j,:]
  row_sum = int(np.sum(row))
  time_with_x_cars_in_sys[row_sum] = time_with_x_cars_in_sys[row_sum] + W_n[j]

#distribution
dist = time_with_x_cars_in_sys/T_n[jumps-1]

column_names = ['fraction_time_in_state']
df = pd.DataFrame(dist, columns=column_names)

# Print the DataFrame
print(df)
```

```
print(df)
```

```
    fraction_time_in_state
0                 0.007294
1                 0.034878
2                 0.083129
3                 0.145419
4                 0.180678
5                 0.183524
6                 0.144695
7                 0.095850
8                 0.064136
9                 0.033028
10                0.015141
11                0.007062
12                0.002490
13                0.001690
14                0.000452
15                0.000202
16                0.000055
17                0.000108
18                0.000089
19                0.000075
20                0.000079
21                0.000047
22                0.000013
```

The long run fraction of time there are no cars in the system.

```
print(df.iloc[[0]])
```

```
    fraction_time_in_state
0                 0.007294
```

The long run average number of customers in the system.

```
avg_cust = 0

for c in range(0,max_customers+1):
  avg_cust = avg_cust + (c*dist[c])

print("Average amount of customers in the system:", avg_cust)
```

```
Average amount of customers in the system: 4.926091380233136
```

Any other features of the long run distribution of the number of cars in the system you're interested in.

Lets look at the longrun distribution of cars between the queues.

```
max_queue_length = int(data.max())
time_with_x_cars_in_q = np.zeros([max_queue_length+1,queues])

for j in range(0,jumps):
  for q in range(0,queues):
    cars = int(data[j,q])
    time_with_x_cars_in_q[cars,q] = time_with_x_cars_in_q[cars,q] + W_n[j]

#distribution
dist_queues = time_with_x_cars_in_q/T_n[jumps-1]

column_names = ['Q1','Q2','Q3','Q4','Q5','Q6',]
df = pd.DataFrame(dist_queues, columns=column_names)

# Print the DataFrame
print(df)
```

```
         Q1        Q2        Q3        Q4        Q5        Q6
0  0.306137  0.312939  0.313201  0.320218  0.306765  0.320262
1  0.562342  0.566067  0.556834  0.552325  0.559145  0.555879
2  0.126766  0.116499  0.122819  0.120562  0.127687  0.120345
3  0.004889  0.004453  0.006294  0.006759  0.005926  0.003585
4  0.000000  0.000177  0.000987  0.000270  0.000611  0.000063
```

Above we can see how the distribution between the queues over time is about the same.

The long run distribution of the amount of time a customer spends in the system.

The average time a customer spends in the system.

Any other features of the long run distribution of the amount of time a customer spends in the system you're interested in.

Part 2

Lets find out how the distribution across queues change when there is a larger portion of the population preferring one orientation over another. Let's see if that affect business.

```
import numpy as np

# Variables to describe the system
pumps_total = 12
pumps_queue = 2
queues = 6
rate_arrival_total = 1
rate_service_pump = 0.2
rate_service_total = 0.2 * pumps_total

ra_lambda = rate_arrival_total

rs_mu = rate_service_total

#Rates are per minute and we are simulating for a week
min_in_week = 60*24*7

n_jumps = 30000

X_t = np.zeros([n_jumps, queues])
W_n = np.zeros(n_jumps)
T_n = np.zeros(n_jumps)

for n in range(1,n_jumps):
    #generate value from exp dist
    Y = np.random.exponential(scale=1)

    #determine cars at pumps
    serving_cars = 0
    for q in range(0,queues):
      serving_cars = serving_cars +  min(pumps_queue, X_t[n - 1,q])

    #determine waiting time and total elapsed time
    roos_total = ra_lambda + rate_service_pump * serving_cars # roos = rate out of state
    W_n[n - 1] = Y / roos_total
    T_n[n] = T_n[n - 1] + W_n[n - 1]

    #determine if event is a service or arrival
    #determine probabilities
    p_arrival = ra_lambda/roos_total
    p_service_1 = rate_service_pump*min(pumps_queue, X_t[n - 1,1-1])/roos_total
    p_service_2 = rate_service_pump*min(pumps_queue, X_t[n - 1,2-1])/roos_total
    p_service_3 = rate_service_pump*min(pumps_queue, X_t[n - 1,3-1])/roos_total
    p_service_4 = rate_service_pump*min(pumps_queue, X_t[n - 1,4-1])/roos_total
    p_service_5 = rate_service_pump*min(pumps_queue, X_t[n - 1,5-1])/roos_total
    p_service_6 = rate_service_pump*min(pumps_queue, X_t[n - 1,6-1])/roos_total
    p_event = [p_service_1, p_service_2, p_service_3, p_service_4, p_service_5, p_service_6, p_arrival]

    p_total = sum(p_event)
    # if p_total != 1:
    #   print("prob total error")
    #   print(p_total)

    #generate random event
    event = np.random.choice([0, 1, 2, 3, 4, 5, 6], replace=True, p=p_event)

    #check if arrival or service and adjust X accordingly
    if event == 6:
      #print("arrival")
      #arrival - determine if orientation preference and decide on queue
      side = np.random.choice(["left", "right", "either"], replace=True, p=[0.6, 0.1, 0.3])

      line = np.zeros(0)
      queue_selection = -1

      if side == "left":
        #print("left")
        left_queues = [X_t[n - 1,1-1], X_t[n - 1,3-1], X_t[n - 1,5-1]]
        left_min = min(left_queues)

        for i in range(0,3):
          if left_queues[i]==left_min:
            line = np.append(line,i)


        queue_selection = int(np.random.choice(line))*2
```

```
      elif side == "right":
# Find the index where T_n exceeds min_in_week
first_index_exceeding_min = np.argmax(T_n > min_in_week)

# print(T_n[first_index_exceeding_min-1] > min_in_week)
# print(T_n[first_index_exceeding_min] > min_in_week)
# print(T_n[first_index_exceeding_min+1] > min_in_week)

print("A week is reached in T_n at index:", first_index_exceeding_min)
```

```
     A week is reached in T_n at index: 19946
```

```
# cut data to longer than week time span (should be big though)
import pandas as pd

if (first_index_exceeding_min != 0):
  data = X_t[0:first_index_exceeding_min,:]
  jumps = first_index_exceeding_min
else:
  data = X_t
  jumps = n_jumps


row_sums = np.sum(data, axis=1)
max_customers = int(max(row_sums))
time_with_x_cars_in_sys = np.zeros(max_customers+1)

for j in range(0,jumps):
  row = data[j,:]
  row_sum = int(np.sum(row))
  time_with_x_cars_in_sys[row_sum] = time_with_x_cars_in_sys[row_sum] + W_n[j]

#distribution
dist = time_with_x_cars_in_sys/T_n[jumps-1]

column_names = ['fraction_time_in_state']
df = pd.DataFrame(dist, columns=column_names)

# Print the DataFrame
print(df)
```