**The simulation of a queue at a petrol station**

Coursework Assignment 2 for the Course “MN-2015 Programming for Analytics”

School of Management

Swansea, December 11th, 2017

**Table of Contents**

[1 Running the simulation of a queue 4](#_Toc500688205)

[2 Calculating the length of a queue 7](#_Toc500688206)

[3 Analysing the simulation results 10](#_Toc500688207)

[4 Management report 12](#_Toc500688208)

# Running the simulation of a queue

The management of a petrol station is concerned with increasing its sales. It is suggested that the operation of the petrol station can be analysed by simulating the customers’ single-server queue. We estimate that during open hours approximately 139 customers (180 minutes/1.3 minutes) would arrive on average. However, we can simulate the queue for 150 customers and discard those who arrive after closing time. So, we set i=150, where i is the number of customers in a queue throughout the 3-hour period.

The queueing process that these customers undergo can be decomposed into the events as follows:

Arrive – time the customer i arrives;

Between – time between the arrivals of customer i and i+1;

Arrive=arrive+between – time the customer i+1 arrives;

Serve – time that takes to serve a customer. This time on average is higher after 9.30 pm because getting served at a hatch takes longer than in the shop.

Start – time a customer starts to get served, Start=max(arrive, finish). We take the highest value between the customer arrival time and the finish time for the previous customer because we deal with a single-server line, and a customer cannot get served before it is not finished with the previous one.

Similarly, Finish – time when a customer is finished to get served and leaves the queue, finish=start+serve. For the first customer, finish=arrive+serve, and for other customers finish=max(arrive,finish)+serve.

Queue – time a customer spends queueing, queue = start-arrive; when start=arrive, queue = 0, i.e. no queueing.

We specify two types of customers: type 1 – those who arrive before 9.30 pm and get served inside the station; type 2 – those who arrive after 9.30 pm and get served at the hatch. We can imagine that the petrol station would have both an entrance and an exit. After 9.30 pm the entrance would just stop letting people inside, and they would have to use the hatch. Meanwhile, those who are already inside could just walk out of the exit door.

The following SAS code produces the desired queue simulation throughout the whole 3-hour period:

**data** line;

retain arrive **0**;

retain finish **0**;

do i=**1** to **147**; /\* to be adjusted to 146 \*/

rand=ranuni(**1**);

between=**1.3**\*ranexp(**1**);

arrive=arrive+between;

if arrive<**90** then type=**1**;

if arrive>=**90** then type=**2**;

if type=**1** then serve=**1**\*ranexp(**1**);

if type=**2** then serve=**1.55**\*ranexp(**1**);

start=max(arrive,finish);

finish=start+serve;

queue=start-arrive;

total=queue+serve;

output line;

end;

**proc** **report** data=line nofs;

columns i type between serve arrive finish start queue total;

define i/display;

define type/display;

define between/display format=**10.5**;

define serve/display format=**10.5**;

define arrive/display format=**10.5**;

define finish/display format=**10.5**;

define start/display format=**10.5**;

define queue/display format=**10.5**;

define total/display format=**10.5**;

**run**;

with the following output:

Table 1 Queue Simulation

| **i** | **Type** | **between** | **Serve** | **Arrive** | **Finish** | **Start** | **Queue** | **Total** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 0.03948 | 0.91673 | 0.03948 | 0.95621 | 0.03948 | 0.00000 | 0.91673 |
| 2 | 1 | 0.10613 | 0.03120 | 0.14561 | 0.98741 | 0.95621 | 0.81060 | 0.84180 |
| 3  …. | 1  ….. | 0.82120  ….. | 2.99986  ….. | 0.96681  ….. | 3.98727  ….. | 0.98741  ….. | 0.02060  ….. | 3.02046  ….. |
| 69 | 1 | 1.70708 | 0.04498 | 89.01703 | 90.12727 | 90.08229 | 1.06525 | 1.11023 |
| 70  …. | 2  …. | 1.69300  …. | 0.90728  …. | 90.71003  …. | 91.61731  ….. | 90.71003  ….. | 0.00000  ….. | 0.90728  ….. |
| 146 | 2 | 0.28632 | 1.20427 | 176.49790 | 221.93376 | 220.72949 | 44.23159 | 45.43586 |
| 147 | 2 | 3.78091 | 0.88390 | 180.27881 | 222.81765 | 221.93376 | 41.65495 | 42.53885 |

Note that we adjusted the customer number from 150 to **146** after running the simulation for the first time and finding that 146 customers would arrive before 11 pm (arrive=176.4979). Customer 147 arrives already after 11 pm (arrive=180.278881). Along with the service times for each customer, we also calculated the variable ‘total’ which is the total waiting times. These would include both the time a customer spends in line waiting to get served and the service time. It can be calculated either ‘finish-arrive’ or ‘queue+serve’. It is worth noting that customers 1 and 70 are served as soon as they arrive, so their total=serve.

These customers are the ones who arrive first under the service regimes mentioned before, so they do not have to queue. At the beginning there is no line because the petrol station just opened. Then, at the time of transition, customer 70 would not be let inside. Therefore, customer 70 would be the first in the new queue outside. We set up the station operation in such a way that the arrival before 9.30 pm is the sufficient criteria to get served under the first regime. So, customers i<70 could still enjoy the first regime, with lower service times, and customers 70<=i<=146 would have to use the hatch. Customer 147 would not be served at all because he/she arrived after 11 pm. In reality, customers who arrive closer to the end would not join the queue, no matter what size it is. They would see that the station is closing at 11 pm because the station worker would put the notice at the hatch window.

# Calculating the length of a queue

The variables generated in the previous output are sufficient to calculate the length of a queue. The variable ‘length’ specifies how many customers are standing in the queue, including those who are being served. To calculate this value, we create a dataset, with both arrivals and departures for each customer in the same column and in chronological order:

**data** arrive;

set line;

evtype=**1**;

events=arrive;

keep type evtype events;

output;

**data** finish;

set line;

evtype=**2**;

events=finish;

keep type evtype events;

output;

**data** events;

set arrive finish;

**proc** **sort** data=events;

by events;

**proc** **print** data=events;

**run**;

The part of the produced output is shown in Table 2.

Table 2 Dataset ‘Events’

| **Obs** | **Type** | **evtype** | **events** |
| --- | --- | --- | --- |
| **1** | 1 | 1 | 0.039 |
| **2** | 1 | 1 | 0.146 |
| **3** | 1 | 2 | 0.956 |
| **4** | 1 | 1 | 0.967 |
| **5**  **…** | 1  … | 2  ….. | 0.987  ….. |

‘Arrival’ is marked as 1 and ‘departure’ is marked as 2 in the ‘evtype’ column. Each of these events change the queue length, with ‘arrival’ increasing the queue by 1 and ‘departure’ decreasing it by 1. Since the length of the queue would only be relevant for the person who just arrives and decides if he/she wants to join it, we only output the cumulated length for ‘arrival’. Note that our method uses the assumption that the arrived person would join the queue, no matter how long it is (hence, length=length+1). We are aware that this is somewhat unrealistic assumption and a serious limitation to the simulation presented. However, it allows us to approximate the potential queue sizes, given certain service times.

Note that we also keep variable ‘type’ to divide the ‘events’ dataset afterwards. It makes sense because the customers who arrive after 9.30 pm start queueing outside the petrol station. So, there are line 1 before 9.30 and line 2 after 9.30 for which the lengths shall be calculated separately:

**data** length1;

set events;

if type=**1**;

retain length -**1**;

if evtype=**2** then length=length-**1**;

else length=length+**1**;

if evtype=**1** then output;

keep length;

**data** length2;

set events;

if type=**2**;

retain length -**1**;

if evtype=**2** then length=length-**1**;

else length=length+**1**;

if evtype=**1** then output;

keep length;

**proc** **print** data=length1;

**proc** **print** data=length2;

**run**;

with the datasets ‘length1’ and ‘length2’ as respective queue sizes:

Table 3 Dataset ‘length1’

Table 4 Dataset ‘length2’

| **Obs** | **Length** |  |  |  |  | **Obs** | **Length** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | 0 |  |  |  |  | **1** | 0 |
| **2** | 1 |  |  |  |  | **2** | 1 |
| **3** | 1 |  |  |  |  | **3** | 1 |
| **4**  **….** | 1  ….. |  |  |  |  | **4**  **….** | 2  ….. |
| **66** | 2 |  |  |  |  | **74** | 24 |
| **67** | 3 |  |  |  |  | **75** | 25 |
| **68** | 2 |  |  |  |  | **76** | 25 |
| **69** | 1 |  |  |  |  | **77** | 26 |

So, the variable ‘length’ represents how many customers are at the petrol station when a new customer arrives, including both a customer being served now and the new one.

# Analysing the simulation results

As said before, 69 customers managed to arrive before 9.30 pm. In the model, these customers were joining the first line inside the shop. We calculated the minimum, maximum and mean of the both queues as well as the minimum, maximum and mean of the respective waiting times with the ‘proc means’ procedure:

**data** line1;

set line;

if type=**1**;

merge length1;

**proc** **means** data=line1 min max mean;

var length total;

**data** line2;

set line;

if type=**2**;

merge length2;

**proc** **means** data=line2 min max mean;

var length total;

**run**;

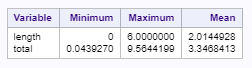
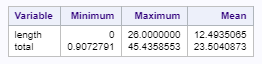
The output as follows:

Table 6 ‘Line2’, ‘at the hatch’

Table 5 ‘Line1’, ‘inside the shop’

As shown in Table 5, the length of the line inside the shop and, hence, waiting times stay in a decent range, with respective means 2.015 and 3.35 minutes. The service is fast enough not to let the line go out of control. It is likely that people would join the line of such a size and our model assumption would be valid. However, if we increase service times a little from mean 1 to 1.3 minute and run a simulation again, we see that such a line would queue outside the shop, with the maximum queue size of 16 people (see Table 7). You can imagine that nobody would queue in such a way.

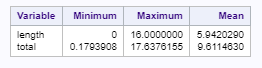


Table 7 ‘Line 1’, ‘inside the shop’, service time = 1.3

Table 6 shows that the second queue at some point started to increase exponentially. So did the waiting times for the people in this line. In real world, you would expect that people would not queue at all. However, our assumption that new customers would still join the line allows us to see the effect of the slower service time on the queue accumulation.

To test a simulation with other service times (as shown above), we introduced both service times as macro variables or parameters 1 and 2 into our model:

%let par1=**1**;

%let par2=**0.97**;

**data** line;

retain arrive **0**;

retain finish **0**;

do i=**1** to **146**;

rand=ranuni(**1**);

between=**1.3**\*ranexp(**1**);

arrive=arrive+between;

if arrive<**90** then type=**1**;

if arrive>=**90** then type=**2**;

if type=**1** then serve=&par1\*ranexp(**1**);

if type=**2** then serve=&par2\*ranexp(**1**);

….(the rest of the program)

# Management report

To analyse the operation of the petrol station, we ran a queue simulation. We based our calculations on the probability distributions for the customers’ inter-arrival times and service times drawn from the historical data. The distributions are exponential, with mean 1.3 minutes for inter-arrival times, 1 and 1.55 minutes for service times inside the shop and at the hatch respectively. We simulated the queues for two operating regimes separately and calculated the average and maximum queue lengths and waiting times for both (see Tables 5 and 6).

Our assumptions:

1. the customer getting served stands in a queue;
2. service time is included in the total waiting time;
3. each new customer joins the line, no matter how long the line is at the time of arrival.

Obviously, the last assumption is not realistic, and that is what you can see from the averages and maximums for the second line. You would not expect that people would still join the line of 12 people and create a line of 26 people. Nobody would wait for 23.5 minutes to get petrol at our station, but they would go to our competitors. So, these people can be translated into the company losses. At the point the queue goes out of control, we lose customers.

Our simulation shows that under the first regime we control the line, and the average waiting time stays under the acceptable 4 minutes. However, the simulated line after 9.30 pm captures the effect of the higher service time. At some point the queue starts growing exponentially, and so do the waiting times. As shown in Table 6, the maximum of the simulated queue reaches 26 customers. In the simulation, the queue went out of control which means that we most probably lost customers in reality.

So, it is recommended to improve the service speed at the hatch. It is easy to assume that it should reach the levels of the in-shop service. To test it, we reran our simulation, with different values for the hatch service time. We found out that the service time should be reduced by 0.58 minutes at minimum (from mean 1.55 to 0.97) to keep mean waiting time less than 4 minutes and, therefore, keep our customers. The new averages and maximums are shown in Table 8.

Table 8 ‘Line 2’, ‘at the hatch’, service time = 0.97



We think that new credit card technology at the hatch could aid solving this problem. It must be tested on several of our petrol stations if this solution would work.