Introducing Queuing Theory Through Simulations

Soon Wan Mei
e-mail: wanmei.soon@nie.edu.sg
Mathematics and Mathematics Education Academic Group,
National Institute of Education, Nanyang Technological University, Singapore

Ang Keng Cheng
e-mail: kengcheng.ang@nie.edu.sg
Mathematics and Mathematics Education Academic Group,
National Institute of Education, Nanyang Technological University, Singapore

Abstract

Queuing theory is usually introduced to students from second year onwards in a university undergraduate programme, as the mathematical principles governing queues can be fairly demanding, making it challenging to introduce any earlier. However, we often see queues and experience queuing in real life. It would therefore be appropriate, relevant and useful to introduce the concept of queuing theory to pre-university students or first-year undergraduates. The approach suggested is through simulation models supported by suitable technology. In doing so, students can understand some basic probability theory and statistical concepts, such as the Poisson process and exponential distribution, and learn how queues may be modelled through simulation, without the need to know all about classical queuing theory. In this paper, we will discuss the role that simulation can play in a classroom to create real world learning experiences for students. To provide a concrete illustration, a set of real data collected in a simple ATM queue will be used to explain how students can systematically be engaged in a modelling activity involving queues. Following that, queues at cinema ticketing counters are studied to discuss the modelling of a more complex queue system.

1. Introduction

Queuing is part of our everyday life. For example, we queue at the checkout counters at supermarkets, for banking services in a bank and to purchase food at fast-food restaurants. A queue forms whenever demand exceeds the existing capacity to serve. This real-life phenomenon, though commonly seen, is not usually studied in pre-university or even first-year undergraduate courses. *Queuing theory* may be incorporated into undergraduate Operations Research or Statistics courses at more senior levels due to its complexity and the demand for mathematical maturity of students. If discussions of queues were done at all at lower levels, it would not usually involve working through the whole process, such as including the collection of real data.

In this paper, we propose that queues be taught to pre-university students or first-year undergraduates, who need not understand all the details of the theory, but can still appreciate a real application of mathematics through simulations. Though some researchers have discussed the use of simulation in teaching mathematics to high school or university students (see for example, Goldsman [1], Reed [2] and Sánchez [3]), they either do not consider modelling or do not present their teaching processes in detail. Our focus is on showcasing how the entire modelling process of the mechanism of queues, from data collection to constructing simulation of queues (not simply using a black box) and to analysis of the queuing model may be introduced to students.

Ang [4] discussed the importance of promoting mathematical modelling in classroom practices and the use of technology as a bridge for the cognitive gap that hinders a student from carrying out a modelling task. The advantages of using simulation as a pedagogical device has been discussed widely (see [1], [2], [3] and [5]). Teachers indicated their beliefs in the usefulness of simulation activities in solving problems, giving meaning to and enhancing the understanding of concepts [3]. Students often find active participation in simulation to be more interesting, intrinsically motivating and closer to real-world experiences than other learning modes [5].

Hence in a nutshell, we are proposing the use of simulation and modelling to teach either students who do not have enough background in mathematics and probability theory and need bridging to high level queuing concepts, or students who may not be inclined to proceed to high level mathematics, but can appreciate queues which are a part of life in the modern world. In the process, students are exposed to a whole package of mathematical knowledge; modelling processes, stochastic processes, mechanisms of queues, and real applications.

2. Basics of an M/M/1 queue

In this section, we will present some basics of queuing theory that instructors may wish to discuss with their pre-university or first-level university students to enable them to partake in the whole modelling process of queues. In particular, we focus on teaching modelling of queues using the M/M/1 model. For simplicity of exposition and convenience of data collection, queues at an automatic teller machine (ATM) are considered. Figure 1 shows a schematic of a typical queue at an ATM. Note that an ATM can also be more reliable in terms of service times since it is not “human” and will not get tired over time.

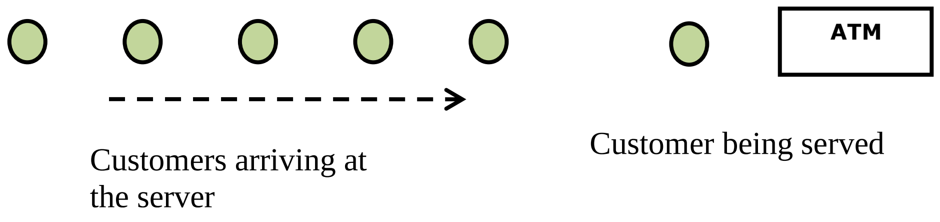


Figure 1: *A single-server queue at an ATM*

In a simple queuing model, the three components involved are the *arrival process, service process* and the *queue structure.* The arrival process typically involves three aspects:

- how customers arrive, for example, singly or in groups (batch or bulk arrivals)
- how arrivals are distributed in time, for example, what is the distribution of **inter-arrival times** (times between successive arrivals)
- whether the number of customers is finite or infinite.

The characteristics of a service process include the following:

- how long the service will take, that is, the **service time distribution**
- number of servers available
- whether each server has a separate queue or there is one queue for all servers.

A queue structure determines:

- how a person is chosen to be served from a set of waiting customers, for example, first-in first-out (or first-come first-served), last-in first-out or randomly
- whether there is balking (customers do not join queue if it is too long), reneging (customers leave queue after waiting for too long), jockeying (customers switching between queues)
- if the queue is of finite or infinite capacity.

In an ATM queue, customers arrive randomly over time and wait for their turns in a single queue, and the ATM (single-server) serves one customer at a time on a “first in first out” basis. The modelling task is to construct a model that can simulate such a queuing system.

The simplest and most commonly considered queue is the M/M/1 model, where the "1" implies that there is only one server. The first "M" stands for Markov or memoryless and means arrivals occur according to a Poisson process. A Poisson process is a stochastic process (a collection of random variables used to represent the evolution of a system over time) where the inter-arrival times are exponentially distributed. That is, if a represents the average number of customers arriving per unit time, then the probability that the inter-arrival time T exceeds the value t is given by $P(T > t) = e^{-at}$.

The second "M" also stands for Markov and denotes that service times of the server are exponentially distributed. That is, if b represents the average number of customers served per unit time, then the probability that the service time S exceeds the number s is given by $P(S > s) = e^{-bs}$. We assume that both the inter-arrival and service times follow exponential distributions because this distribution is the only continuous distribution that possesses the unique *memoryless* property. That is, the probability of waiting an additional time unit for the next customer arrival does not depend on how long it has been since the previous arrival, and the probability of completing a service within the next given time period is independent of how long the person has been served already.

For the simple queuing system above, there are useful formulae that can be derived under the assumption that the system has reached a **steady state -** that is, the system has been running long enough so as to settle down into some kind of equilibrium position. Or in other words, the operating characteristics of the queue (for example, expected waiting time and expected number of customers in the system) do not vary with time. But note of course that in real-life, systems often do not reach such a state.

Let $\rho = \frac{a}{b}$ be the traffic intensity, that is, a measure of traffic congestion for the server. It is clear that if $\rho < 1$, that is, the average arrival rate is less than the average service rate, then the queue length approaches a constant and the system reaches steady state. Otherwise, the queue grows indefinitely. Then according to the M/M/1 model, the expected steady state waiting time in a queue W_q and the expected total time spent in system W are

$$W_q = \frac{\rho}{b-a} \qquad (1)$$

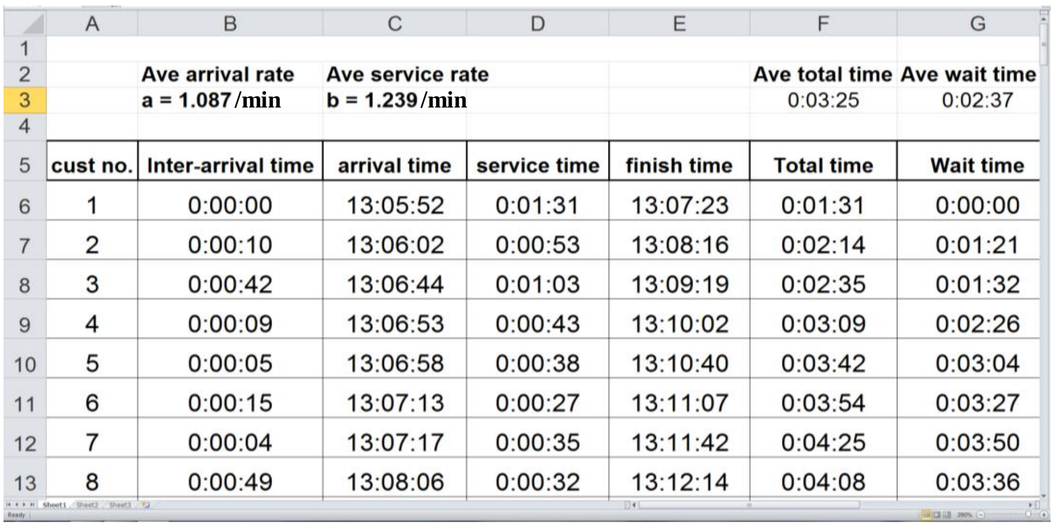
and

$$W = \frac{1}{b-a}. \qquad (2)$$

Though the derivations for the formulae above are complex in classical queuing theory, these equations are actually simple and easy to use. That is, given the arrival and service rates, we can easily calculate the expected times W_q and W in a queuing system. Note that the above and other relevant formulae (e.g., for average queue length) can be derived using *birth-death processes*. Students who are more mathematically mature can be referred to Hillier and Lieberman [6] or Bunday [7] for details.

3. Data collection and simulation process of an ATM queue

With a basic understanding of queuing theory, students can proceed to collect data at an ATM to obtain estimates of the important parameters a and b. For example, students can video-record a queue or use a digital watch to record the arrival and finish times of customers on site. The inter-arrival times, service times, wait times and total times (wait time and service time) can then be calculated with the aid of an electronic spreadsheet. As an example, Figure 2 shows a screenshot of an MS Excel worksheet with data that we obtained from observing an ATM queue for approximately an hour at our university campus around late lunch time (when there is sufficient traffic flow).



	A	B	C	D	E	F	G
1							
2		Ave arrival rate	Ave service rate			Ave total time	Ave wait time
3		a = 1.087/min	b = 1.239/min			0:03:25	0:02:37
4							
5	cust no.	Inter-arrival time	arrival time	service time	finish time	Total time	Wait time
6	1	0:00:00	13:05:52	0:01:31	13:07:23	0:01:31	0:00:00
7	2	0:00:10	13:06:02	0:00:53	13:08:16	0:02:14	0:01:21
8	3	0:00:42	13:06:44	0:01:03	13:09:19	0:02:35	0:01:32
9	4	0:00:09	13:06:53	0:00:43	13:10:02	0:03:09	0:02:26
10	5	0:00:05	13:06:58	0:00:38	13:10:40	0:03:42	0:03:04
11	6	0:00:15	13:07:13	0:00:27	13:11:07	0:03:54	0:03:27
12	7	0:00:04	13:07:17	0:00:35	13:11:42	0:04:25	0:03:50
13	8	0:00:49	13:08:06	0:00:32	13:12:14	0:04:08	0:03:36

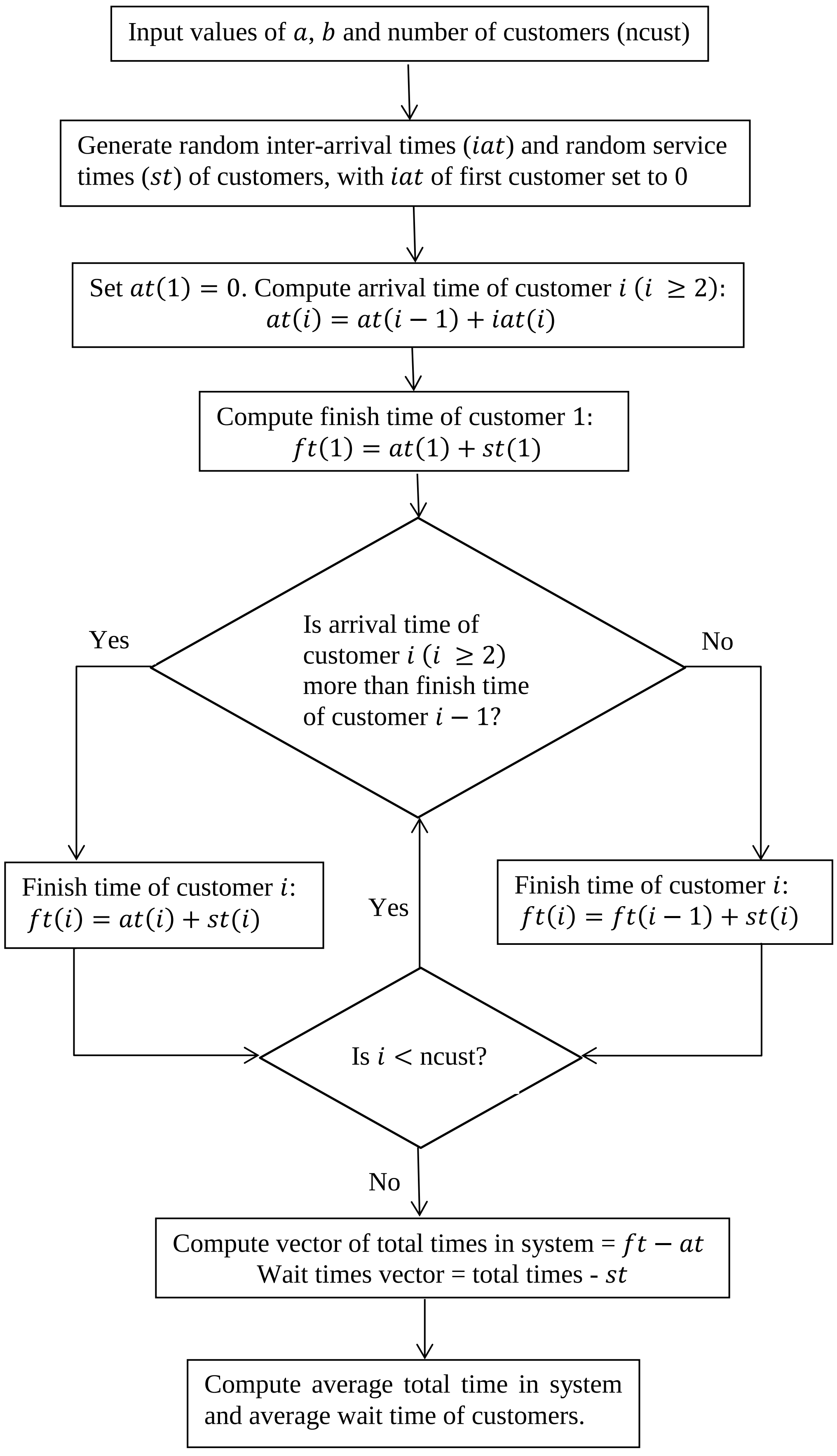
Figure 2: *Screenshot of Excel worksheet with data from an ATM queue*

In this case, the inter-arrival times in column B are obtained from the arrival times in column C (e.g., cell B7 = C7-C6). The service times in column D are calculated using columns C and E (e.g., cell D7 = E7-MAX(E6,C7)). That is, the service time of a customer depends on whether the customer arrives before/after the previous customer has completed the service. It is clear that total times in column F are the differences between the finish and arrival times, while the wait times in column G are the differences between total times and service times. Note that $a = \frac{\text{no. of customers}}{\text{sum (inter-arrival times)}}$ and $b = \frac{\text{no. of customers}}{\text{sum (service times)}}$. The values of a, b, average total and average wait times can then be calculated easily and are reflected in Figure 2.

Students can better appreciate the mechanism of queues and learn about generating randomness through simulation. Instructors can help students to understand the steps of each simulation run using a flowchart as shown in Figure 3. Note that we assume the inter-arrival time and arrival time of the first customer to be 0 (for convenience).

The random inter-arrival times (iat) and service times (st) can be generated using "Inverse Transform Method". For example, to generate random numbers from an exponential distribution with parameter a, we first generate r uniform random numbers over [0,1]. Then let $r = 1 - e^{-at}$ (cumulative distribution function). It follows that $t = -\frac{1}{a}\log(1-r)$. Since $r \sim U(0,1)$, we have $1 - r \sim U(0,1)$. Thus, we can simply let $t = -\frac{1}{a}\log(r)$. That is, we set $iat = -\frac{1}{a}\log(r)$. Similarly, we set $st = -\frac{1}{b}\log(r')$, where r' are also uniform random numbers generated over [0,1]. For more details, refer to [8].

To execute the simulation program, the user will also need to input the number of simulation runs desired, that is, we are actually doing Monte Carlo simulation. The idea is to calculate results many times, each time using a different set of random values from the uniform distribution. The belief is that averaging the results of many simulations should provide a better indication of real behaviour. The output of the whole program will then be the average total time in the system and average wait time of customers, taken over *all* simulation runs. In addition, the standard deviations of the average total time and wait time of the simulations are computed. More details are provided as comments within the program provided in Appendix A.



Input values of a, b and number of customers (ncust)
Generate random inter-arrival times (iat) and random service times (st) of customers, with iat of first customer set to 0
Set $at(1) = 0$. Compute arrival time of customer i ($i \geq 2$): $at(i) = at(i-1) + iat(i)$
Compute finish time of customer 1: $ft(1) = at(1) + st(1)$
Is arrival time of customer i ($i \geq 2$) more than finish time of customer $i-1$?
Yes
No
Finish time of customer i: $ft(i) = at(i) + st(i)$
Finish time of customer i: $ft(i) = ft(i-1) + st(i)$
Yes
Is $i <$ ncust?
No
Compute vector of total times in system = $ft - at$
Wait times vector = total times - st
Compute average total time in system and average wait time of customers.

Figure 3: *Flowchart describing a simulation run of the ATM queue*

4. Analysis of results for an ATM queue

We will use the values of a, b and number of customers observed from one of our data sets to discuss how simulation can be used to verify real data or relate to classical theory. With input values $= 1.087$, $b = 1.239$, ncust =75 and nsim = 1×10^6, the program output were "ave wait time = 3.057 mins" and "ave total time = 3.865 mins". The standard deviations of the average wait time and average total time from simulations were 2.246 mins and 2.306 mins respectively.

Table 4: *Summary of different average times for an ATM queue*

Average times	**Simulated**	**Real data**	**Theoretical**
Wait time	3.057 mins	2.62 mins	5.772 mins
Total time	3.865 mins	3.42 mins	6.579 mins

The table above summarizes all the different average wait and total times obtained for this queue. Note that the theoretical times are calculated using equations (1) and (2). We see that these values vary quite a lot from the simulated and real times. They fall just outside one standard deviation from the simulated average times. The discrepancies could be due to the small sample size (75 customers) and probably the short duration (about 70 mins) of the experiment. If traffic flow could be monitored over several days, perhaps more accurate results could be obtained.

However, we can check that the actual average wait time and total time fall within half a standard deviation from the simulated average values. Thus perhaps we can conclude that M/M/1 model is quite suitable to model ATM queues.

Up to this point, our focus is on a single-queue-single-server system. For a single-queue-multi-server system, the theory involved, the data collection process and the simulation procedure are undoubtedly more complex. However, it is still possible to introduce such a queue to students through modelling and simulation.

5. Basics of an M/M/c queue

To introduce an M/M/c queue, instructors basically only need to discuss the underlying components of a single-queue-multi-server system, similarly to that discussed in Section 2. As before, the arrival process, service process and the queue structure constitute such a system. For simplicity, we will use a queue with 3 ticketing counters at a Cineplex as an example. Note that the two "Ms" in M/M/c are the same as before, and the "c" (where $c > 1$) refers to the multiple number of servers. Here, we have $c = 3$. See Figure 5 below.

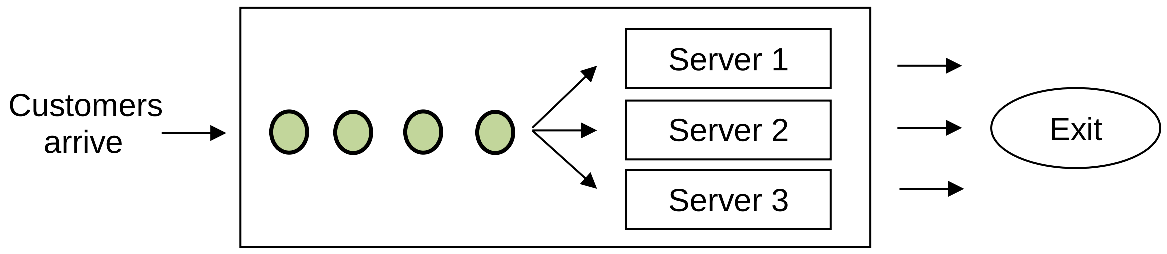


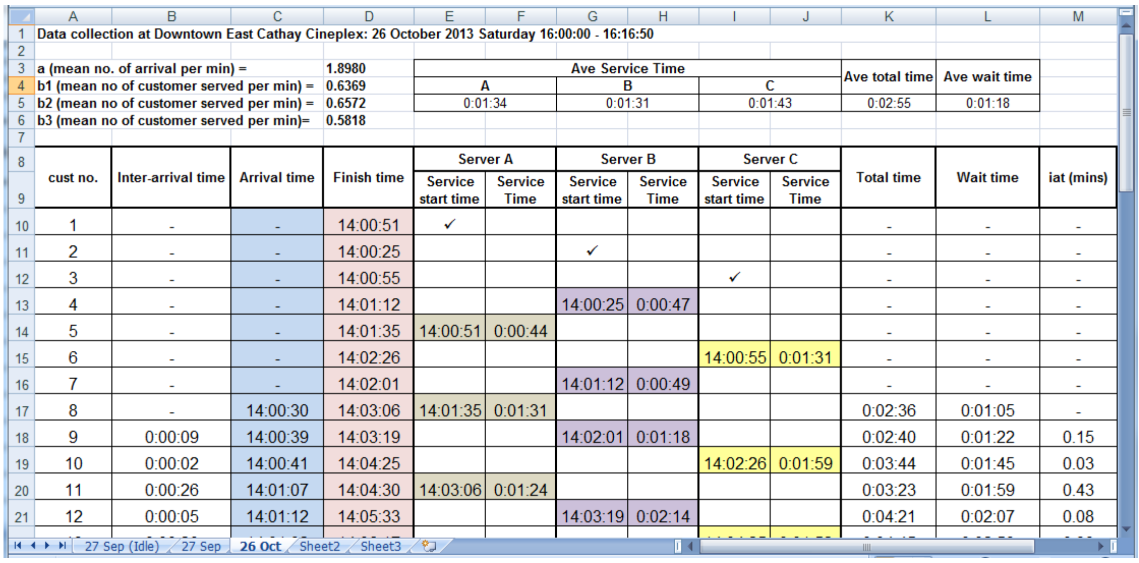
Figure 5: *A multi-server queue at a Cineplex*

The main difference between this multi-server ticketing queue and the single-server ATM queue is that though customers are attended to in a first-come first-served manner, a customer who arrives first may not leave the system first, as it depends on the service time. Similar to the M/M/1 case, there are formulae for the expected steady state waiting time in a queue and the expected total time spent in a single-queue-multi-server system, assuming that $\rho < 1$ and same service rate for all the servers. However, the expressions are a lot more complicated and may not be easy for students at these levels to grasp. Hence we will omit them here.

6. Data collection and simulation process of a multi-server ticketing queue

Data was collected downtown at a Cineplex with three servers (named Server A, Server B and Server C) using a video camera. The arrival times, finish times and service start times of the customers, along with which servers they used, were then extracted from the video and recorded in an excel spreadsheet as shown in Figure 6. Each customer's service time was found from the difference between the finish time and the service start time. The average service time for each of the three servers was then easily obtained. Other relevant times were calculated similarly as before.

Note that the queue data was recorded during a busy Saturday afternoon, hence it was difficult to wait for the system to be empty (i.e., no customer being served or in queue), to start with the collection of data. The data of 39 customers was recorded in a time span of 17 minutes. At the start of data collection, there were three customers being served and four customers who were in the queue. Hence the arrival times of the first 7 customers were unknown. When the collection of data ended, there were 3 customers who were still being served and 5 other customers who were still in the queue. Thus the finish times of these 8 customers were not recorded. In addition, it was observed that customers randomly approach any idle server if more than one were available.



Data collection at Downtown East Cathay Cineplex: 26 October 2013 Saturday 16:00:00 - 16:16:50

a (mean no. of arrival per min) =	1.8980
b1 (mean no of customer served per min) =	0.6369
b2 (mean no of customer served per min) =	0.6572
b3 (mean no of customer served per min)=	0.5818

Ave Service Time			Ave total time	Ave wait time
A	B	C		
0:01:34	0:01:31	0:01:43	0:02:55	0:01:18

cust no.	Inter-arrival time	Arrival time	Finish time	Server A		Server B		Server C		Total time	Wait time	iat (mins)
				Service start time	Service Time	Service start time	Service Time	Service start time	Service Time			
1	-	-	14:00:51	✓						-	-	-
2	-	-	14:00:25			✓				-	-	-
3	-	-	14:00:55					✓		-	-	-
4	-	-	14:01:12			14:00:25	0:00:47			-	-	-
5	-	-	14:01:35	14:00:51	0:00:44					-	-	-
6	-	-	14:02:26					14:00:55	0:01:31	-	-	-
7	-	-	14:02:01			14:01:12	0:00:49			-	-	-
8	-	14:00:30	14:03:06	14:01:35	0:01:31					0:02:36	0:01:05	-
9	0:00:09	14:00:39	14:03:19			14:02:01	0:01:18			0:02:40	0:01:22	0.15
10	0:00:02	14:00:41	14:04:25					14:02:26	0:01:59	0:03:44	0:01:45	0.03
11	0:00:26	14:01:07	14:04:30	14:03:06	0:01:24					0:03:23	0:01:59	0.43
12	0:00:05	14:01:12	14:05:33			14:03:19	0:02:14			0:04:21	0:02:07	0.08

Figure 6: *Screenshot of excel data of a Cineplex ticketing queue system*

The average arrival rate a and average service rates for the three servers b_1, b_2 and b_3 respectively, can be calculated as before and used as inputs for our single-queue multi-server simulation program, together with the number of customers and number of simulations required. Figure 7 shows the flow of each simulation run in a nutshell. As before, we generate the random inter-arrival times using $iat = -\frac{1}{a}\log(r)$, where $r \sim U(0,1)$.

The main differences between this simulation program and that used for M/M/1 queues are as follows:

- Different random service times are generated for the three servers based on the different average service rates, that is, for each server i, we set $st_i = -\frac{1}{b_i}\log(r')$, where $r' \sim U(0,1)$.
- Additional matrices are used to record the actual service time and server used by each customer, frequencies of use of the servers, and times when the servers are available

- We have to identify idle servers and choose randomly among idle servers to service next customer

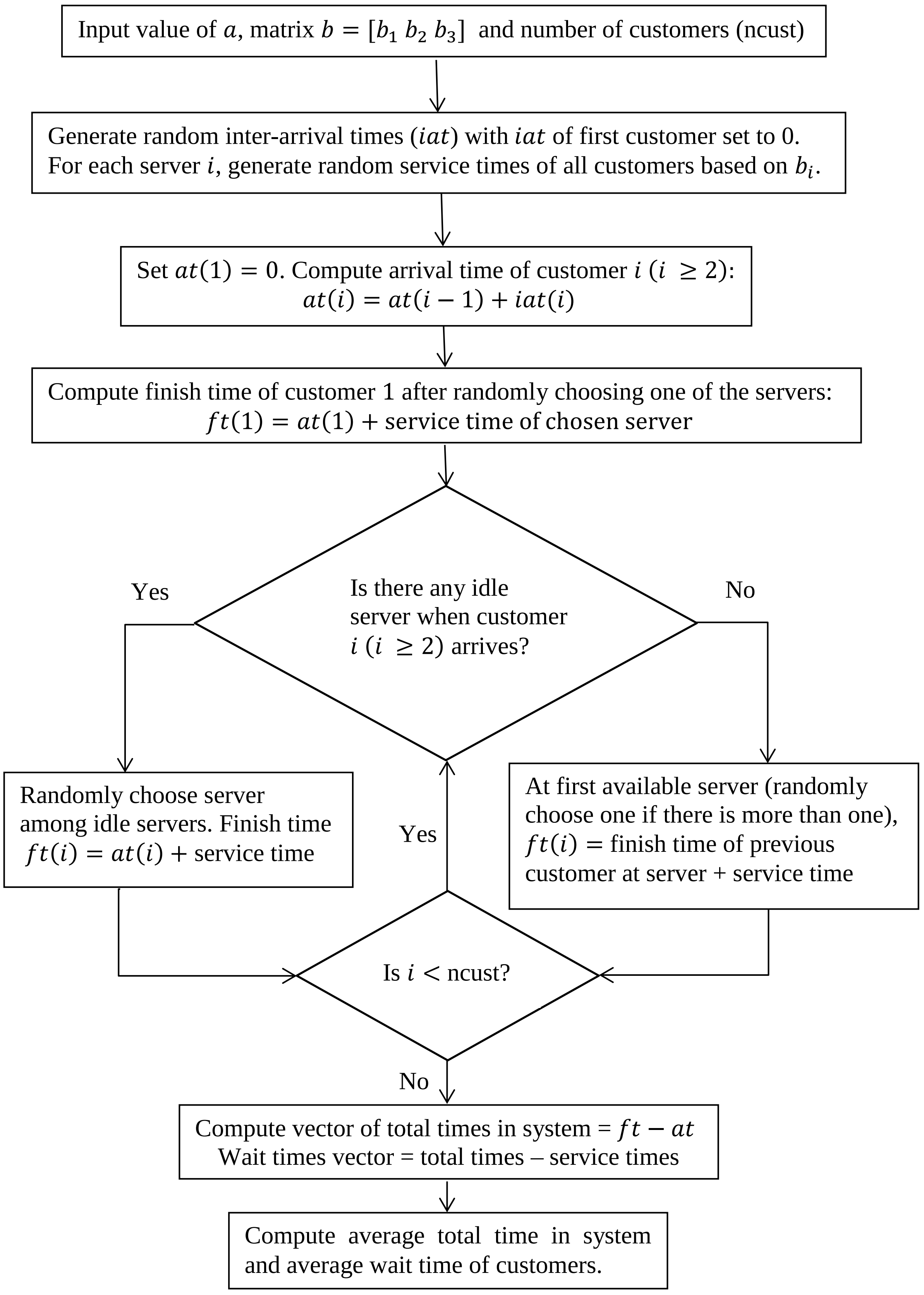

Input value of a, matrix $b = [b_1\ b_2\ b_3]$ and number of customers (ncust)
Generate random inter-arrival times (iat) with iat of first customer set to 0. For each server i, generate random service times of all customers based on b_i.
Set $at(1) = 0$. Compute arrival time of customer i ($i \geq 2$): $at(i) = at(i-1) + iat(i)$
Compute finish time of customer 1 after randomly choosing one of the servers: $ft(1) = at(1) +$ service time of chosen server
Is there any idle server when customer i ($i \geq 2$) arrives?
Yes
No
Randomly choose server among idle servers. Finish time $ft(i) = at(i) +$ service time
At first available server (randomly choose one if there is more than one), $ft(i) =$ finish time of previous customer at server + service time
Yes
Is $i <$ ncust?
No
Compute vector of total times in system = $ft - at$
Wait times vector = total times – service times
Compute average total time in system and average wait time of customers.

Figure 7: *Flowchart describing a simulation run of a Cineplex ticketing queue*

The detailed purposes of different parts of the program are provided as comments within the code given in Appendix B.

7. Analysis of results for a multi-server ticketing queue

As depicted in Figure 6, the "actual average wait time was 1 minute 18 seconds or 1.3 minutes", and the "average total time was 2 minute 55 seconds or 2.917 minutes". With input values of $a = 1.8980$, $b_1 = 0.6369$, $b_2 = 0.6572$, $b_3 = 0.5818$, ncust = 39 and nsim = 50000, the simulation program output were "ave wait time = 2.2559" and "ave total time = 3.8788". The standard deviations of the average wait time and average total time from simulations were 2.2281 mins and 2.5858 mins respectively.

Though the actual times are not close to the simulated times, it is possible that with a greater sample size, a longer duration of the experiment within a day, and observation of the queue over several days, results may be more accurate. In addition, we can see that the actual average wait time and total time fall within half a standard deviation from the simulated times. Hence perhaps an M/M/c model may still be quite suitable to model a cinema ticketing queue system.

8. Conclusions

In this paper, we present an approach to teach the modelling of single-queue-single-server and single-queue-multi-server systems through simulation to students who may not be mathematically mature enough to understand classical queuing theory. Through the entire process, students can learn a great variety of concepts while appreciating the natural phenomenon of queues in real life. Our hope is that pre-university and undergraduate educators will find this work useful in teaching statistics, modelling, and real-life applications of mathematics. As an extension, we can introduce the process of a multi-queue-multi-server system to students through modelling and simulation in a similar manner.

9. References

[1] D. Goldsman, *A simulation course for high school students*, in *Proceedings of the 2007 Winter Simulation Conference,* S.Henderson, B. Biller. M. Hsieh, J. Shortle, J. Tew, and R. Barton, eds., J.W. Marriott Hotel, Washington, D.C. 9-12 December, 2007, pp. 2353-2356.

[2] J.H. Reed, *Computer simulation: A tool to teach queuing theory*, Experiential Learning Enters the Eighties 7 (1980), pp. 63-66.

[3] E.S. Sánchez, *Teachers' beliefs about usefulness of simulation with the educational software fathom for developing probability concepts in statistics classroom***,** in *Proceedings of The Sixth International Conference on Teaching Statistics*, B. Phillips, ed., Holiday Inn, Cape Town, South Africa, 7-12 July, 2002

[4] K.C. Ang, *Teaching and Learning Mathematical Modelling with Technology,* in *Proceedings of the 15th Asian Technology Conference in Mathematics,* W.C. Yang,M. Majewski, T. Alwis, and P.H. Wooi, eds., Kuala Lumpur, Malaysia, 17-21 December, 2010, pp. 19-29.

[5] L.M. Lunce, *Simulations: Bringing the benefits of situated learning to the traditional classroom*, Journal of Applied Educational Technology 3(1) (2006), pp. 37-45.

[6] F.S. Hillier and G.J. Lieberman, *Introduction to operations research,* McGraw-Hill College, New York, 2001.

[7] B.D. Bunday, *An introduction to queuing theory,* Oxford University Press, Oxford, England, 1996.

[8] L. Devroye, *Non-Uniform Random Variate Generation*, Springer-Verlag, New York, 1986.

10. Appendices

Appendix A

```
% Simple queuing theory simulation, M/M/1 queue
a = input('Input the mean number of arrivals per minute:');
b = input('Input the mean number of customers served per minute:');
ncust = input('Input the number of customers:');
nsim = input('Input the number of simulations required:');

%Initialise ave_wait_time_matrix, ave_total_time_matrix
ave_wait_time_matrix = [];
% to initialize matrix of average wait times for all simulations
ave_total_time_matrix = [];
% to initialize matrix of average total times for all simulations

for k = 1:nsim   % to run “nsim” number of simulations

        % Notations:
        % at = arrival time of a person joining the queue
        % st = service time (the time spent at the ATM machine)
        % ft = finish time after waiting and being served.
        %
        % initialize arrays:
        at = zeros(ncust,1); % all arrival times are initialized
        ft = zeros(ncust,1); % all finish times are initialized

        % Generate random arrival times assuming Poisson process:
        r = rand(ncust-1,1); % generate “ncust-1” uniform random numbers
        iat = -1/a * log(r); % generate inter-arrival times according to
                             % exponential distribution
        iat = [0; iat]; %to set zero iat of first customer
        at(1) = 0; % arrival time of first customer is assumed 0

        for i=2:ncust
                at(i) = at(i-1) + iat(i); % arrival times of other customers
        end

        % Generate random service times for each customer:
        r = rand(ncust,1); % generate “ncust” uniform random numbers
        st = -1/b * log(r); % generate service times according to
                            % exponential distribution

        % Compute time at which each customer finishes:
        ft(1) = at(1)+st(1); % finish time for first customer

        for i=2:ncust
                ft(i) = max(at(i)+st(i), ft(i-1)+st(i));
                % to obtain finish time for all other customers
        end
        total_time = ft - at; % total time spent (in queue and in service)
        wait_time = total_time - st; % time spent waiting in queue

        for j = 1:ncust
                if wait_time(j) < 0
                      wait_time(j) = 0; % to manually set wait time to be zero
                                        % when there are computer errors
                end
        end

        ave_wait_time = sum(wait_time)/ncust; % compute average wait time
        ave_total_time= sum(total_time)/ncust; %compute average total time
        ave_wait_time_matrix = [ave_wait_time_matrix; ave_wait_time];
        %to add on ave_wait_time of current simulation to matrix
        ave_total_time_matrix = [ave_total_time_matrix; ave_total_time];
        %to add on ave_total_time of current simulation to matrix

end

ave_wait_time_final = sum(ave_wait_time_matrix)/nsim
% to find average wait time taken over all simulations
ave_total_time_final =  sum(ave_total_time_matrix)/nsim
% to find average total time taken over all simulations

sq_dev_wait_time
= (ave_wait_time_matrix -ave_wait_time_final*ones(nsim,1))
  .*(ave_wait_time_matrix - ave_wait_time_final*ones(nsim,1));
% to find square of deviations of average wait time of each simulation from average
% wait time take over all simulations

std_dev_wait_time = sqrt(sum(sq_dev_wait_time)/(nsim-1))
% to find standard deviation of wait time

sq_dev_total_time
= (ave_total_time_matrix-ave_total_time_final*ones(nsim,1))
 .*(ave_total_time_matrix - ave_total_time_final*ones(nsim,1));
% to find square of deviations of average total time of each simulation from average
% total time take over all simulations

std_dev_total_time = sqrt(sum(sq_dev_total_time)/(nsim-1))
% to find standard deviation of total time
```

Appendix B

```
% Simple queuing theory simulation, M/M/c queue
% Multiple servers, single queue:
a = input('Input the mean number of arrivals per minute:');
b = input('Input the matrix of average service rates for the multiple servers:');
ncust = input('Input the number of customers:');
nsim = input('Input the number of simulations required:');

%Identify number of servers
nserv = length(b);

%Initialise ave_wait_time_matrix, ave_total_time_matrix
ave_wait_time_matrix = [];      % to initialize matrix of average wait times
                                 % for all simulations
ave_total_time_matrix = [];     % to initialize matrix of average total times
                                % for all simulations

for k = 1:nsim

    % Notations for each simulation run:
    % at = arrival time of a person joining the queue
    % ft = finish time after waiting and being served
    % sn = to record which server serves each customer
    % st = record of actual service time for each customer
    % fserv = records freq of cust served by each server
    % mst = matrix of service times for all possible customers at all servers
    % rec = matrix to keep record of the times servers can be available

    % initialize arrays:
    at = zeros(ncust,1);
    ft = zeros(ncust,1);
    sn = zeros(ncust,1);
    st = zeros(ncust,1);
    fserv = zeros(1,nserv);
    mst = zeros(ncust,nserv);
    rec = zeros(ncust,nserv);

    % Generate random arrival times of customers assuming Poisson process:
    r = rand(ncust-1,1);  % generate “ncust-1” uniform random numbers
    iat = -1/a * log(r);  % generate inter-arrival times according to
                          % exponential distribution
    iat = [0; iat];       %to set zero iat of first customer
    at(1) = 0;    % arrival time of first customer is assumed 0

    for i=2:ncust
        at(i) = at(i-1) + iat(i); % arrival times of 2nd to last customers
    end
```

```
% Generate random service times for each customer at each server:
r = rand(ncust,1);  %generate random numbers from uniform distribution

for i=1:nserv
    mst(:,i) = -1/b(i)*log(r);
end

% Compute finish time of each customer, and update sn, st, fserv and rec:
% for customer 1
s = randi([1,nserv],1);  % choose randomly among all servers
ft(1) = at(1) + mst(1,s);  % ft is according to service time of server "s"
sn(1) = s;  % update that cust 1 is served by server "s"
st(1) = mst(1,s);  % update service time of customer 1
fserv(s) = 1;  % update that server "s" has served one customer
rec(1,s) = ft(1);  % update record of time server "s" is available

% for all other customers
 for i = 2:ncust

   rec(i,:) = rec(i,:)+rec(i-1,:);  % update record before ith customer arrives
   m = min(rec(i-1,:));  % find earliest available time among all servers
   minIDX = [];  % initialise servers that can serve ith customer

   if at(i) < m  % ith customer needs to wait

      for j = 1:nserv
         if rec(i-1,j) == m
            minIDX = [minIDX j]; %add on servers that can serve ith customer
         end
      end

      L = length(minIDX); %no. of servers to choose from
      r = randi([1,L],1); %choose integer randomly from 1 to L.
      s = minIDX(r); %locate which server corresponds to chosen "r"
      %for ft(i), need to add free time of serv s to service time of
      %customer next to be served by serv "s"
      ft(i) = rec(i-1,s)+mst(fserv(s)+1,s);

   else  % ith customer does not need to wait

      for j = 1:nserv
         if rec(i-1,j) <= at(i)
            minIDX = [minIDX j];  % add on servers that can serve ith customer
         end
      end

      L = length(minIDX);  % no. of servers to choose from
      r = randi([1,L],1);  % choose integer randomly from 1 to L.
      s = minIDX(r);  % locate which server corresponds to chosen "r"
      %for ft(i), we add arrival time of customer i to service time of
      %customer next to be served by serv "s"
      ft(i) = at(i)+mst(fserv(s)+1,s);
   end
```

```
        sn(i) = s; % update that customer i is served by server "s"
         st(i) = mst(fserv(s)+1,s);  % update service time of customer i
         fserv(s) = fserv(s)+1;  % update no. of customer served by "s"
         rec(i,s) = ft(i);  % update record after ith customer arrives and is served

      end

      % Generate other statistics
      total_time = ft - at;  % total time spent by each customer
      wait_time = total_time - st;  % time spent waiting before being served

      for j = 1:ncust
         if  wait_time(j) < 0
            wait_time(j) = 0;   % to manually set wait time to be zero
                                % when there are computer errors
         end
      end

      ave_wait_time = sum(wait_time)/ncust;
      ave_total_time = sum(total_time)/ncust;
      ave_wait_time_matrix = [ave_wait_time_matrix; ave_wait_time];
      %to add on ave_wait_time of current simulation to matrix
      ave_total_time_matrix = [ave_total_time_matrix; ave_total_time];
      %to add on ave_total_time of current simulation to matrix

end

ave_wait_time_final = sum(ave_wait_time_matrix)/nsim
% to find average wait time taken over all simulations
ave_total_time_final =  sum(ave_total_time_matrix)/nsim
% to find average total time taken over all simulations

sq_dev_wait_time = (ave_wait_time_matrix - ave_wait_time_final*ones(nsim,1)).*
(ave_wait_time_matrix - ave_wait_time_final*ones(nsim,1));
% to find square of deviations of average wait time of each simulation from average wait
% time take over all simulations

std_dev_wait_time = sqrt(sum(sq_dev_wait_time)/(nsim-1))
% to find standard deviation of wait time

sq_dev_total_time = (ave_total_time_matrix - ave_total_time_final*ones(nsim,1)).*
(ave_total_time_matrix - ave_total_time_final*ones(nsim,1));
% to find square of deviations of average total time of each simulation
% from average total time take over all simulations

std_dev_total_time = sqrt(sum(sq_dev_total_time)/(nsim-1))
% to find standard deviation of total time
```