USING RUN TESTS TO ELIMINATE INITIAL UNSTEADY STATE DURING SIMULATION

by

H.GOPALKRISHNA MENON

B.Sc (Met. Engg.), Sambalpur University, 1984

A REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

INDUSTRIAL ENGINEERING

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1988

Approved by:

[illegible]

Major Professor

ACKNOWLEDGEMENT

The author wishes to express his sincere appreciation and gratitude to Dr. L.E.Grosh for his valuable guidance and encouragement throughout this project and his study in this University. The author also wishes to thank his Major Professor once again for being of assistance during extremely trying circumstances faced during the course of the program.

The author is also indebted to Dr. Doris Grosh for serving on the graduate committee and has been of assistance as the head of the department. The author also wishes to thank Dr. Mark McNulty for serving on the graduate committee.

Table of Contents

Section I

Introduction

1.1.0 <u>Purpose of study</u>

To develop and provide a signal for the end of the initial transient state of a simulation so that data collection may begin.

1.2.0 <u>Introduction</u>

There are three phases in the investigation by simulation that take place after the problem has been identified and the model formulated:

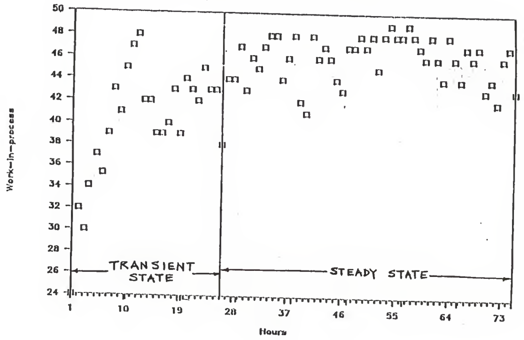
1. Model implementation - description in a language acceptable to the appropriate computer.
2. Strategic planning - design of an experiment that will yield the desired information.
3. Tactical planning - determination of how each of the test runs specified in the experimental design is to be executed.

These experiments deal with phase three, which is probably less important than phase 1 or 2, but is nevertheless significant. Using extremely large sample sizes can overwelm virtually all the difficult tactical questions, but this is not a practical or good solution. We must not forget that computer time is not a free good [Conway, 1954].

1.2.1 <u>The Problem of Initial Transients</u>

This report specifically deals with a problem which is

Figure 1-1 Work-in-Process vs time



present in every simulation. This is the question of equilibrium: since it generally takes some time for the simulation model to "warm up", performance measurement should not begin until after this period. At this stage it may be necessary to define what we mean by initial transient and steady state. We define steady state here as:

> **"Steady state is said to be reached when successive observations in a simulation experiment are statistically indistinguishable" [Emshoff/Sisson, p.189, 1970].**

We take this to imply that the data model is

$$x_i = mean + e_i$$

where the mean is a constant,

e_i an error term with a mean of zero.

We only require that the central limit theorem apply and allow us to use the normal inferences for hypothesis testing. When the experiments are statistically distinguishable transient phase is said to exist. The effect of the warm up period is shown in Figure 1-1 which should not be considered in the data collection period. Each measurement in a simulation is a chance variable and the investigator is obligated to execute his experiment in such a manner that he can estimate the precision of his results. The existing literature on this method show us three methods to tackle the problem of initial transient.

1.2.2 Method I: Exclusion of Initial Data

The first method involves the exclusion of some initial data from consideration. This involves bias to the simulator on the basis of the preconceived notions of the operator. Moreover, it does not form an objective criterion to determine when steady state is attained. The usual method is to take a few pilot runs with periodic collection of data. As a rough guide truncate a series until the first of the series is neither a maximum nor a minimum of the remaining set. The stabilization period is decided by examining the stabilization period for a few runs and thereafter delete the same period from each run afterwards.

The stabilization period must be judged in terms of the number of elements in the system and the number of events that take place. Discarding data for a automobile parts inventory made for a three month period may be adequate. This may not be true for a missile maintenance system where the mean time between failures is six months.

1.2.3 Method II: Setting Initial Conditions

The second method involves setting the initial starting conditions. The most common way of starting a simulation is what might be described as "empty and idle" condition. For most problems this may be inadequate except in case of a single channel queue and the steady state is reached in a relatively short period of time. The proponents of this method argue that by changing conditions and by starting the simulator under certain initial conditions may result in an accelerated equilibrium. But this method also involves the use of a number of pilot runs to determine under what conditions the simulator acheives equilibrium earlier. The probability of the bias to the simulator remain as also the need for long runs. There are also some other problems raised by the use of this method. One of the problem that arises is the choices the investigator has when investigating the problem:

1. Test each system starting "<u>empty and idle</u>".
2. Test each system using a common set of starting conditions that is essentiallly a compromise between different sets of reasonable starting conditions.
3. Test each system under its own "reasonable" starting conditions.

The second strategy is clearly more efficient than the first, since any improvement over the empty and idle model has to be a better choice; the choice between the second and the third is less obvious. The main advantage obtained by the third method is that minimum time is lost in attaining equilibrium. But the problem raised by this method is that there are no existing ways to perform this method except by making pilot runs for each of the conditions that we desire to test. Then, to determine when the equilibrium conditons would prevail, we have to look at the plots of the data collected versus time to decide when the transient ends.

1.2.4 Method III: Using Long Runs

The third and last method proposed by most authors is the use of long runs which may make the bias introduced by the initial starting conditions insignificant. This obviously involves the use of long runs and running the simulator for long periods of time. This is a very inefficient method of performing the simulation experiment. Considerable computer time is lost in the use of long runs. Hence this method should be avoided as far as possible.

1.3.0 <u>Methods of Attack</u>

Numerous simulation problems involve initial transient. The new methods of attack proposed for determining when the initial transient state ends and steady state is reached are described below:

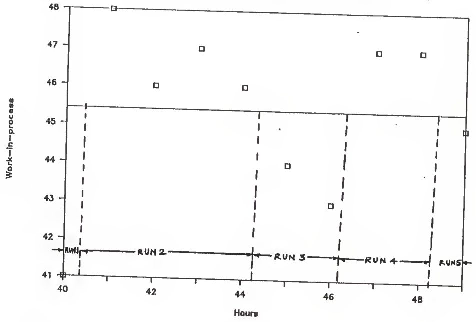
1.3.1 <u>Description of the Methods</u>

Before describing the methods it is necessary to define a run.

> **"A run is described as a succession of similar events preceded and succeeded by different events" [Conover, 1980].**

For our purposes the run would consist of two classes of events which may be above or below the mean. The events above the mean could form runs and the events below the mean could form runs. The number of events in a run is referred to as the <u>length of the run.</u> The <u>total number of the runs</u> and the length of the longest run, and various other statistics can be used to test for randomness of arrangement against the alternative sequence of dependency. This is termed the distribution theory of runs.

Figure 1–2: Showing 5 Runs



1.3.2 Method I: The Wald-Wolfowitz Total Number of Runs Tests

In an ordered sequence of two kinds of observations, the total number of runs may be used as a measure of the randomness of the sequence; too many runs indicate that each observation tends to be followed by an observation of the other type, while too few runs indicate a tendency for like observations to follow like observations. In either case the sequence would indicate that the process generating the sequence was not random. By a random process, or a random sequence, we mean a sequence of independent and identically distributed random variables; that is, a sequence of runs above and below the mean is random if the probability of an above is the same as the probability of a below and the resulting values are independent of each other.

As a test for randomness:

"**DATA:** The data must consist of a sequence of observations, taken in order of occurance. The observations are of two types, or can be reduced to two data types. Let n denote the number of aboves and m denote the number of belows in the observed sequence.

ASSUMPTION: The only assumption for the test of randomness is that the observations be recordable as either type one (above) or type two (below).

HYPOTHESIS: The hypothesis for the test of randomness are:

H_0 : The process which generates the sequence is a random process.

H_1 : The random variables in the sequence are either dependent on other random variables in the sequence, or are distributed differently from one another.

TEST STATISTIC: The test statistic L equals the length of the longest run of like elements in the sequence of observations.

DECISION RULE: In the test for randomness use a two-tailed critical region, and reject H_0 at the level alpha if $L > w_{1-\alpha/2}$ or if $L < w_{\alpha/2}$ [Conover, p.349, 1980].

The distribution function for 20 or more data points is

$$w_p = \frac{2mn}{(m + n)} + 1 + x_p \sqrt{\frac{2mn(2mn - m - n)}{(m + n)^2 (m + n - 1)}}$$

where x_p is the p quantile of a standard normal variable, obtained from Table 1 [Conover, p. 416, 1980]. For smaller number of data points we use the table provided in Conover's Non-Parametric Statistics.

1.3.3 Method II: Length of the longest run as a test for randomness

In a sequence of two kinds of observations, the length of the longest run may be used as a measure of the randomness of the sequence; a long run indicates that a particular type of observation tend to be followed by another ot the same kind. The sequence would indicate that the process generating the sequence was not random. By a random process, or a random sequence, we mean a sequence of independent and identically distributed random variables; that is, a sequence of above and belows is random if the probability of an above is tha same as the probability of a below and the resulting values are independent of each other.

As a test for randomness:

"**DATA:** The data must consist of a sequence of observations, taken in order of occurance. The observations are of two types, or can be reduced to two data types. Let n denote the number of aboves and m denote the number of belows in the observed sequence.

ASSUMPTION: The only assumption for the test of randomness is that the observations be recordable as either type one (above) or type two (below).

HYPOTHESIS: The hypothesis for the test of randomness are:

H_0 : The process which generates the sequence is a random process.

H_1 : The random variables in the sequence are either dependent on other random variables in the sequence, or are distributed differently from one another.

TEST STATISTIC: The test statistic L equals the length of the longest run of like elements in the sequence of observations.

DECISION RULE: In the test for randomness use a one-tailed test critical region, and reject H_0 at the level alpha if L $< w_{1-\alpha/2}$ " [Conover, p. 363, 1980].

1.3.4 <u>Why's a measure of randomness a measure of steady state?</u>

When the process is in transition state, one often observes a trend in the data as the queue length changes from the starting value toward the steady state mean. Once the process has reached steady state, observations will be sufficiently far apart to be nearly independent. This suggests that a test for steady state be conducted by testing for the independence of successive observations. In order to apply this system with the important parameters (utilization ratio and time between observations set at values so that the process exhibits a trend (the transition state) followed by independent values (the steady state).

1.4.0 What's expected during transient state and steady state?

Consider an assembly line of 15 workers in the assembly line processing one part at a time and the time to acheive steady state for the work-in-process is considered. The buffer capacity is kept at a constant value of 4. Also assume that the each of the workers perform with the same ability. The table below gives the task time in seconds followed by the number of people that completed the task:

35 - 0	60 - 240	85 -110	110 - 50	135 - 5
40 - 5	65 - 210	90 - 90	115 - 40	140 - 2
45 - 70	70 - 180	95 - 80	120 - 30	145 - 1
50 - 186	75 - 160	100 - 70	125 - 20	150 - 1
55 - 250	75 - 130	105 - 60	130 - 10	155 - 0

The Assembly line problem can be explained by using the following figure.

```
Source--W --B --W --......--W  --B  --W  --Sink
        1   1   2           14   14   15
        |                             |
        |          System Boundry     |
```

Figure 1-3 Showing the Assembly line system.

The workers in the assembly line have three states:

1. Starved - waiting for a part to enter the previous buffer.

2. Working - they have a part to work on.

3. Blocked - the workers are unable to pass the completed part to the next buffer.

The question that arises is that for a stated uniform buffer capacity how many parts would one expect to be in the system i.e., how many parts would constitute the work in process. The purpose of the buffer in this problem is to decouple the state of worker W_i from W_{i-1} and W_{i+1}. For measuring the contents of the work-in-process the following rules are followed.

1. When the worker W_1 is idle, a part is obtained from the source. Thus a part enters work-in-process inventory.

2. When worker W_{15} completes a part, the part is passed to the sink. Thus a part leaves the work-in-process inventory.

The time trace of the work-in-process is shown in Figure 1-3

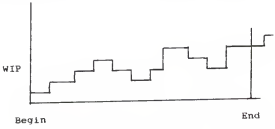


Figure 1-4 Showing change in work-in-process

Figure 1-5: 1st to 10th data points

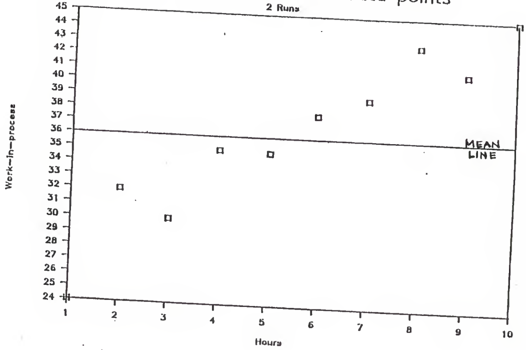
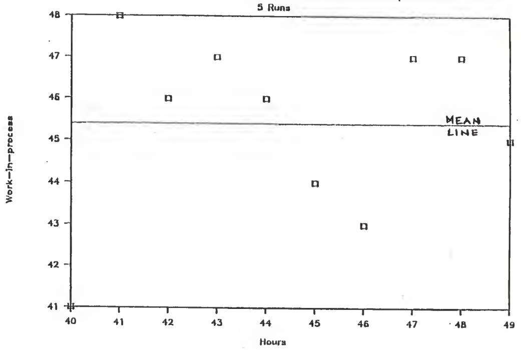


Figure 1-6: 40th to 49th data points



We may measure the average work-in-process by:

1. Averaging the instantaneous values of the Work-in-process observed either at random or fixed time intervals.
2. We may compute the area under the curve for a fixed time interval and divide by the interval length to obtain the average curve height. Then we average these values over the run length.

As the time interval becomes longer, the data takes on the appearance of independent observations. Figures 1-1, 1-5 and 1-6 represent data taken from such a system of instantaneous observations. From the figures shown it may be clearly seen that the total number or runs is less than 4 for the first 10 hours while the length of the run is greater than 4. When examining the total number of runs for hours 40 to 49 we find the length of the longest run to be less than 4 while the total number of runs is greater than 4.

1.4.1 <u>Testing the Effectiveness of the methods</u>

To test the effectiveness of these methods we may set up a simulation problem for which a mathematical solution exists and then try to verify the effectiveness of the method. If the results of this method conform to the other conventional methods then the method could be implemented into the GPSS/H programming language and the method can be used by future users of GPSS/H for determining when steady state begins in other problems. For this the conventional queueing theory problem with a single server with a utilization ratio of 95% was chosen. The various problems that arose out of trying to get a queue with a steady queue length are described in Section II along with the problems of implementation. The use of a single queue with a single server and a 95% utilization rate did not result in a steady state queue length and the various other changes made are described in Section II. The problems faced in implementing into GPSS/H and how they were tackled are also described in Section II.

Section II

Queuing Problem and Applying Theory of runs

2.1.0 Problem Statement:

1. To obtain data for our study, we decided to use an M/M/1 queueing process because theoretical answers are available. We assumed that for a suitably large utilization ratio there would be a transient state building up from zero to the expected queue length with moderate variability.

2. Observations:

   a) Should they be made frequently or far apart. State the observation intervals in terms of the average service time.

   b) Should the observations be:

      i) instantaneous (current content of a storage).

      ii) weighted (average content of a storage during the observation interval).

   c) Should the observations be made:

      i) on a fixed time interval, or

      ii) on a random time interval (Occurance sampling).

3. How does changing the RMULT value affect the experiments. (RMULT is the random number generator seed.)

4. Run test to consider:

   a) Maximum run length in a fixed number of observations.

   b) Distribution of run length in a fixed number of observations.

2.1.1 <u>Analysis of Single Server Queue Problem</u>

There are two types of simulations with regard to analysis of a single system - a terminating simulation or a non terminating simulation. A terminating simulation is one for which the desired measure of system performance are defined relative to the interval of simulated time $[0, T_E]$ where T_E is the instant in the simulation when a specified event occurs. The event E is specified before the simulation begins. A steady state simulation has been defined as one for which the measures of performance are as limits as the simulation goes to infinity. The length of the simulation is made large enough to get good estimates of the quantities of interest [Law/Kelton p.282, 1984].

In this study we try to measure the performance for steady state simulations by means of the example problem stated below:

"Consider the output process of the M/M/1 queue where the length of the queue is to be measured when steady state is attained. This is a single server queue system with the IID exponential interarrival times with the mean $1 / \lambda$. IID exponential service times with the mean $1 / \mu$ and customers are served in a FIFO manner. Assume a $\lambda / \mu < 1$. The object of the non-terminating

simulation of the M/M/1 queue might be to estimate the steady state average length of the queue given some initial condition such as at time 0 the queue length is 0 [Law/Kelton, p284, 1984]".

The steady state length of the queue is calculated by queueing theory where the calculations can be made by the following relationships:

$$\textbf{Expected queue length} = \frac{(\lambda)^2}{\mu(\mu - \lambda)}$$

[Cooper, p.71, 1977]

The calculated queue length is calculated for a 95% utilization rate to be of the order of 18 persons waiting in the queue, while the expected queue length is of the order of 8 persons waiting in the queue for a 90% utilization rate. The queue length decreases as the utilization rate decreases.

2.2.0 Setup of the problem in GPSS/H

The single server queue was setup so that the customer enters the queue and is serviced by the server in FIFO manner. This is shown by the following flow diagram.

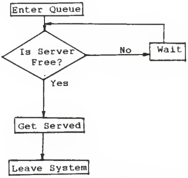


Figure 2-1: Flow Diagram of the Implementation into GPSS/H

The customers are generated in an exponential fashion and we use a random number seed to generate the customers. The instantaneous queue length of the system is collected at time intervals desired by the operator. The collection of average hourly queue length involves colecting the current queue length(X1), current clock time(C1) and the time at which the queue length was different from the current queue length(X2). This is shown in Figure 2.1. The sum of the areas under the curve is computed for every time interval and divided by the total time in the time interval considered. The other variables that can be changed in this simulation besides the data collection interval are the RMULT(Random number generator seed) and the interarrival - service time ratio.

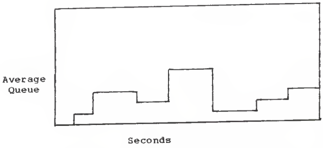


Figure 2-2 Area under the curve shows cumulative Queue Length. If the clock is reset every hour then, dividing cumulative Queue Length by 3600 seconds gives the hourly average queue length.

2.2.1 <u>Should instantaneous or average queue lengths be used</u>:

Based on these calculations a simulation experiment was setup so that we could get an initial transient followed by a steady state. Two experiments were setup to determine whether the observations should be instantaneous or the experiments should consider the average hourly queue length. The simulation experiments programs are shown in Appendix 1 and 2. After running the experiments the results of the experiments were plotted and the results are shown in in figure 2-4 and 2-5. The results show that the such a simulation does not result in transient phase followed by a level phase. The process did not exhibit the properties necessary to make the runs tests for steady state valid. Based on the results obtained it may reasonably be concluded that the instantaneous queue lengths may be discarded because the analysis of a single server queue shows the average queue length to reach a steady state of 18 at 95% utilization. The tests for instantaneous queue lengths were conducted only to determine if instantaneous queue lengths had the possibility of showing properties necessary to make run tests valid.

The effect of not resetting the time integral is shown in Figure 2-6. When the time integral is not reset, we get

Figure 2—3 Average Queue Length

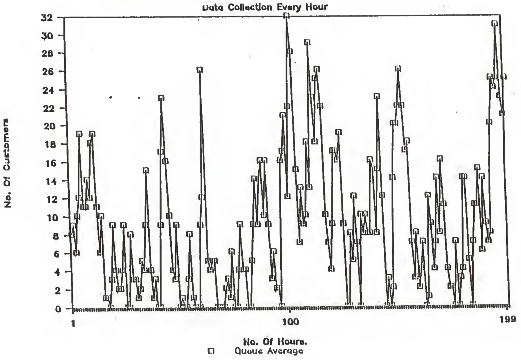


Figure 2–4 Instantaneous Queue Length

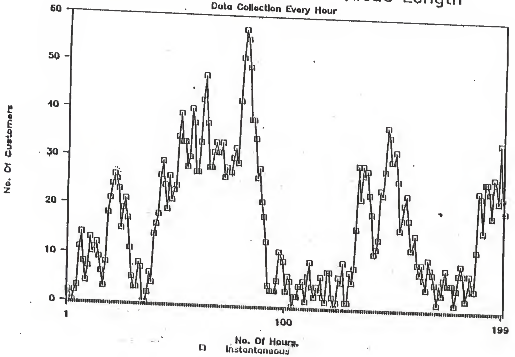
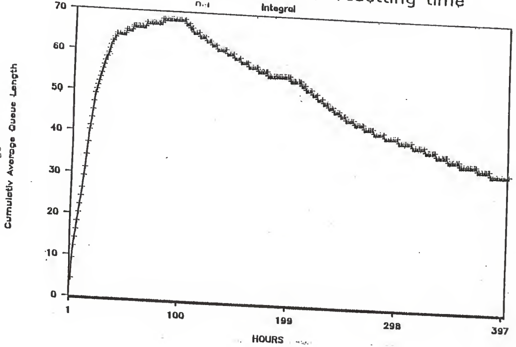


Figure 2--5 Effect of not resetting time



the cumulative average queue length by dividing by the total number of hours. By this we mean that the average queue length goes on adding up and we get the cumulative average queue length by adding average queue length every hour and dividing by the total number of hours. The cumulative sum of the average is shown as the time increases. The other important observation is that the use of random observations are not necessary in view of the fact that the observations do not exhibit the properties necessary to make runs tests for steady state valid.

2.2.2 Effect of changing RMULT

To further examine the length of the steady state queue lengths, it was decided to vary the RMULT values of the program. The resulting queue lengths are shown in Figures 2-6 and 2-7. The results show us that queue lengths are considerably different for for different RMULT values, but does not result in a situation where there is an initial transient followed by steady state. The effect of using different combinations gives the instantaneous queue length and average queue lengths as shown and does not indicate initial transient followed by steady state. The instantaneous queue length is measured at the end of every hour while the average queue length is measured on an hourly basis.

Figure 2—6 Effect of RMULT(431,7)

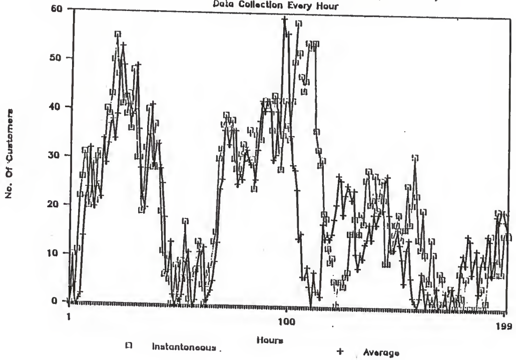
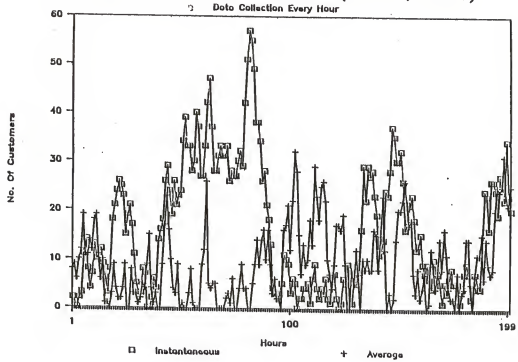


Figure 2--7 Effect of RMULT(43217,23981)



2.2.3 Effect of changing the frequency of observations

The effect of varying the frequency of the observations from 1/2 hour to 1 hour to 2 hours to 10 hours was tried but the results do not indicate any possibility of initial transient followed by steady state as can be seen from changing the Figures 2-8, 2-9, 2-10, 2-11 and 2-12. Changing the frequency of data collection did not result in any kind of steady state and as a result of the considerable variance Theory of Runs could not be applied to determine when the steady state began in the system. This instantaneous queue lengths and the average queue lengths are shown in Figures 2-8, 2-9, 2-10, 2-11 and 2-12. As can be seen clearly there is no indication that steady state would be achieved even though the simulator was run for 200 hours. It may also be observed here that the average queue length lags behind the instantaneous queue length.

Figure 2–8 Varying Data Collection time

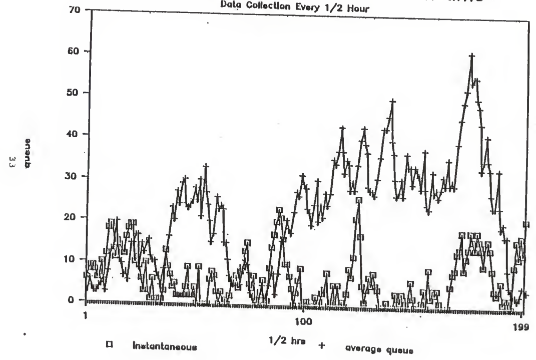


Figure 2-9 Varying Data Collection time

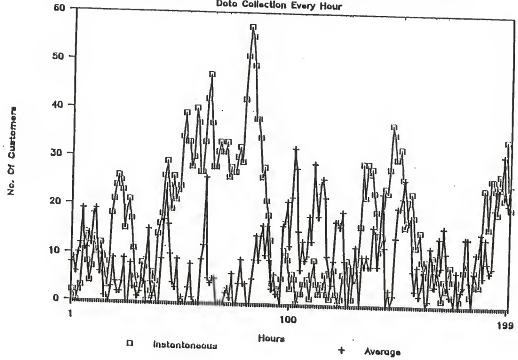


Figure 2-10 Varying Data Collection time

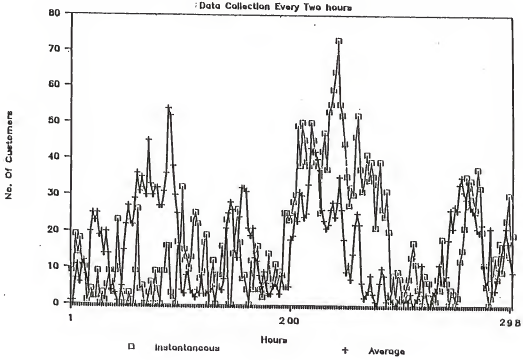


Figure 2-11 Varying Data Collection time

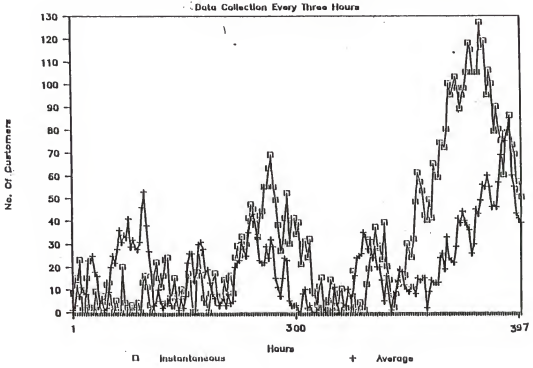
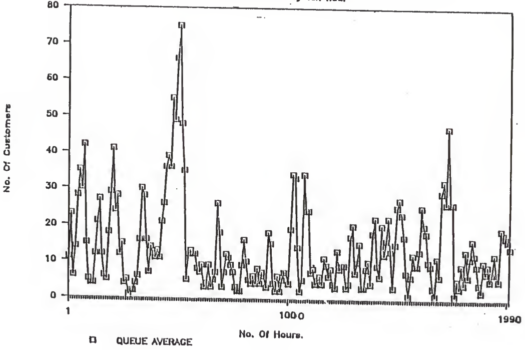


Figure 2-12 Vary Data Collection time

Data Collection Every ten hour



2.2.4 Effect of changing utilization

Based on this it was decided to test the queue lengths when the utilization rate was lowered. The lowering of utilization rate did not result in situation where we got an initial transient followed by steady state. The results of the these simulation runs are shown in figures 2-13, 2-14 and 2-15. The process does not appear to exhibit the properties necessary to make the runs tests for steady state valid.

Figure 2—13 Effect of utilization rate

$\lambda / \mu = 200 / 175$

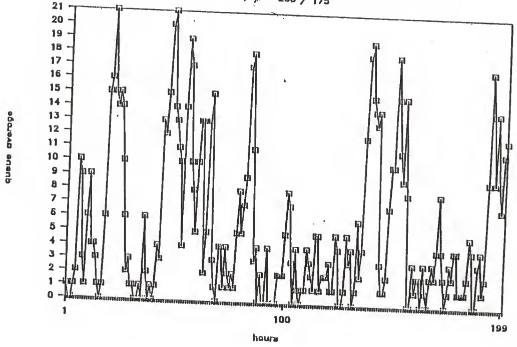


Figure 2—14 Effect of utilization rate

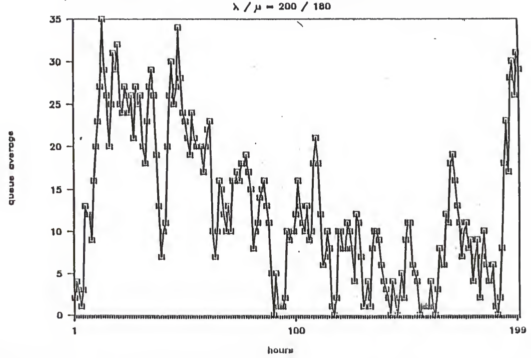
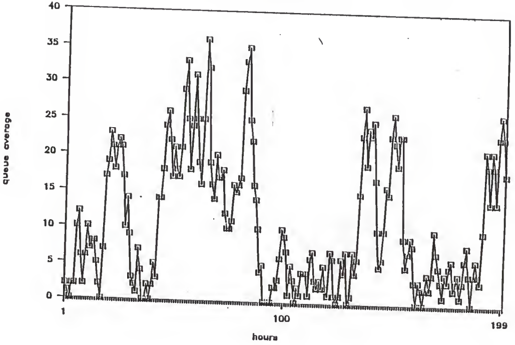


Figure 2–15 Effect of utilization rate

$\lambda / \mu = 200 / 185$



2.2.5 <u>Effect of RMULT</u>

This raised the possiblity that the Random number generators were not creating customers in an exponential fashion and the only alternative was to verify the RMULTS. This was done by using the program as shown in Appendix 5. The results show that the random number generators were not infact generating an exponential distribution. This is shown in figures 2-16 and 2-17.

To test further the interarrival and service distributions and of GPSS/H Version 1.9 were generating an exponential distribution it was found that the new version of GPSS/H called GPSS/H Version 1.9 was infact generating an exponential distribution time and this is shown in figures 2-18 and 2-19. The Chi-Squared goodness of fit test shows a Chi - squared value of 184 for GPSS/H version 1.0 which is greater than the critical value while Version 1.9 gives a Chi-Square value of 14.9 which is less than the critical value. The test had 34 degrees of freedom and the critical value is 18.4. This confirms our suspicion that the distribution is not infact exponential for GPSS/H Version 1.0. However, the use of this method may not be feasible in view of non availablity of literature on the use of Help Blocks in the new version of GPSS/H. The documentation necessary for this has not been made available by the

Figure 2—16 Exponential Inter Arrival

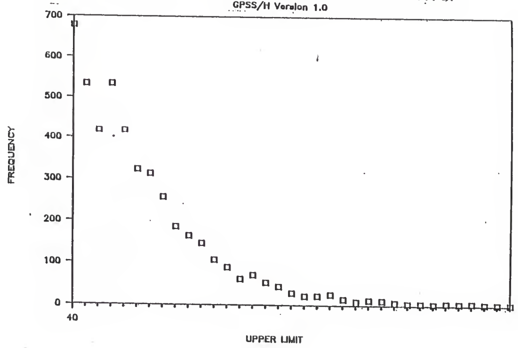


Figure 2—17 Exponential Service time

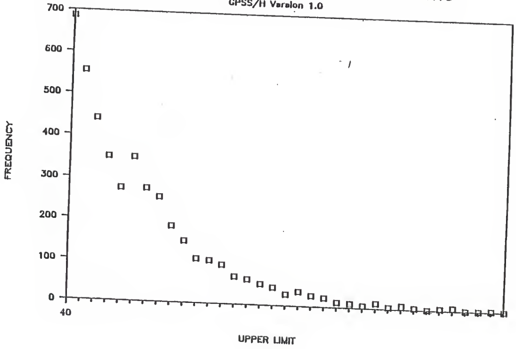


Figure 2—18 Exponential Interarrival

GPSS/H Version 1.9

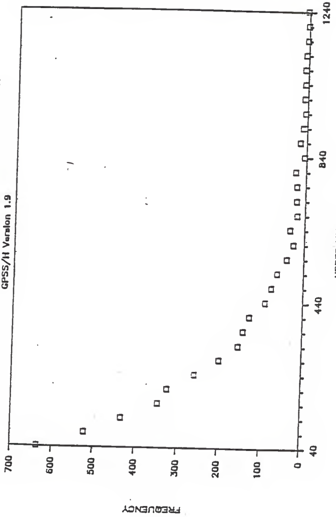
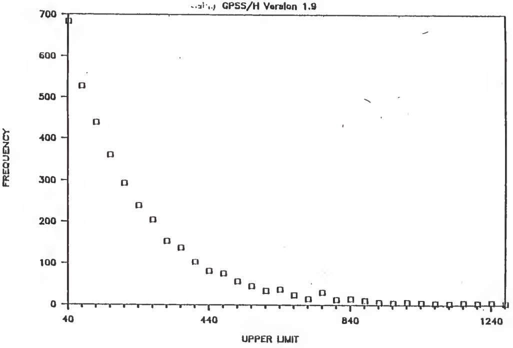


Figure 2—19 Exponential Service Time



Wolverine Software Company. It may be pointed out here that the main difference between GPSS/H Version 1.0 and GPSS/H Version 1.9 is the clock used. Version 1.0 uses an integer clock while the Version 1.9 uses a double precision real clock. This also confirms many tests by Dr. Fishman and Dr. Moore [1986] have proved Lehmer's algorithm for random number generation to be superior to the Taushworthe algorithm used in for random number generation.

2.3.1 Effect of Two Queues

Based on the results of the above experiments it was decided to modify the simulation experiment to have two queues and the queue length of the first queue measured. As the customer is generated he enters the first queue and after he is serviced by the service unit one enters a second queue with the same exponential service time. This was done by using the program as shown in the Appendix 4. The sequence of events is shown in the figure below:

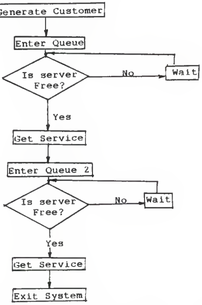


Figure 2-20: Showing the two queue system flow diagram

Figure 2-21 Average First Queue Length

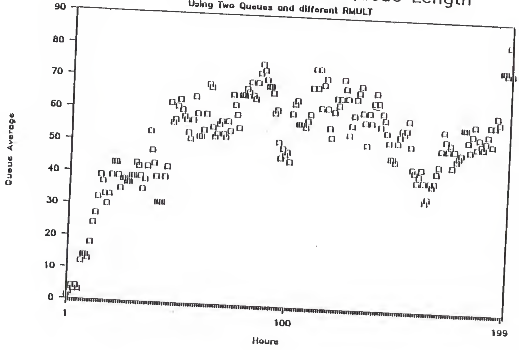
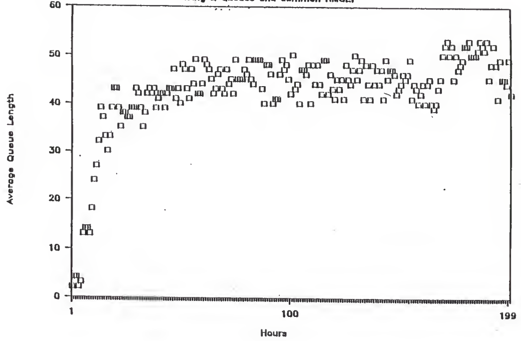


Figure 2—22 Average First Queue length

Using 2 Queues and common RMULT



This results in an average queue length of the first queue shown by the Figure 2-21. It may be noted here that the utilization rate is only 62.5% and the expected average queue length is 2. It appears that the observed average queue length is much higher than the calculated value because sequence of use of the use of the random number seed gets affected by the use of the second queue. The average queue length is used to calculate the moving average and the total number of runs and the length of each run using the program shown in Appendix 5. The use of instantaneous queue lengths has been discarded due to excess variability. This is not suited to our experiment as the process does not exhibit the properties necessary to make the runs tests for steady state valid.

We may note from Figure 2-21 that the process does not exhibit the properties necessary to make the runs tests for steady state valid. So it was proposed to modify the problem by changing the RMULT values used to the same constant number. The result of the change of the Random Number generator is plotted in figure 2-22. The average queue length of the first queue shows that steady state is attained after running the simulator for 28 hours. It should be noted here that the only difference between the one queue simulation and the two queue simulation and the two queue simulation is the sequence in which the random numbers are used.

2.3.2 Applying Theory of Runs

The methods viz., Wald-Wolfowitz total number of runs method and the length of the longest run method can now be applied to the average queue length of the first queue. It may be pointed out here once again that the data is completely random if the data exhibits 4 runs or more and we can conclude steady state to be reached according to the Wald-Wolfowitz theory of longest number of runs. The other method using the length of the longest run requires the longest run to be equal to or shorter than 4. Appendix 10 shows the tables used to determine the total number of runs and the length of longest run for any given number of data points. Using the hourly queue lengths obtained as a result of the experiments can be seen in Table 1 below. Only a part of the data depicting how the system approaches steady state is reproduced. From Table 1 it is clearly seen that the steady state is reached by the 28th hours, since the total number of runs is 4. Subsequent to that the total number of runs is either equal or greater than 4. Similarly using the length of the longest run method we find that the longest run is 4 or less after the 28th hour.

HOURS	MEAN	NO OF RUNS	INDIVIDUAL RUN LENGTHS			
16 25	37.099990	2	6	4		
17 26	37.500000	3	5	4	1	
18 27	38.199990	3	6	2	2	
19 28	39.799980	4	4	4	1	1
20 29	38.799980	4	3	4	1	2
21 30	38.799980	4	2	4	3	1

Table 1: The number of people in the queue is found to reach steady state after 28 hours. The mean of the last 10 readings is 46.3 and the standard deviation is found to be 3.2. This conforms the steady state by the blocking method to be after 28 hours.

The results as verified by running the simulation experiment for 160 hours calculating the mean and the standard deviation and going backwards to determine when steady state begins results in the beginning of steady state at the end of 20 hours. This method is called as the blocking method and is developed by Emshoff and Sisson [1970] and is fairly accurate in determining the end of steady state. The major problem with the use of this method is the need for long runs until we are reasonably sure that the steady state is reached before this method can be applied and more often then not we may have to plot the graph to determine whether steady state is reached. Besides using long pilot runs the other drawback is that the method works backwards and does not determine when transient state ends but when steady state begins. Examining the results obtained by the FORTRAN subroutine shown in Appendix 5 gives us the following results:

1. Steady state begins at the end of 28 hours if we consider using the Wald-Wolfowitz theory of total number of runs.

2. Steady state begins at the end of 28 hours if the length of the longest run is used.

3. The calculations using mean $\pm$ 2 standard deviations indicates that the steady state begins after 28 hours.

2.3.3 Is there a significant difference between the blocks?

For the purpose of testing for a significant differnce between the blocks of ten, an Analysis of Variance was conducted between the various blocks and the results obtained are shown in Table 2. The results indicate a statistically significant difference between the blocks.

	DF	Sum of Squares	Mean Squares	F Value	PR>F
Source	15	12900.8	860.05	76.39	0.0001
Model	144	1621.2	11.25		
Total	159	14522.0			

Table 2: Showing Analysis of Variance Results between blocks

This F-test shows that there is a statistically significant difference between the blocks. To further determine which blocks are different we have to perform the Fisher's LSD test.

Based on the results obtained it was decided to run a test for significant differences between the means of the blocks and by using the Fisher's LSD we get the following results:

Grouping		Mean	N	Observations
A		45.3	10	151 - 160
A		45.3	10	141 - 150
A		45.3	10	131 - 140
A		45.2	10	121 - 130
A		45.2	10	111 - 120
A		45.1	10	101 - 110
A		44.8	10	91 - 100
A		44.7	10	81 - 90
A		44.5	10	71 - 80
A		44.2	10	61 - 70
A		43.8	10	51 - 60
A		42.4	10	41 - 50
A	B	40.4	10	31 - 40
	B	38.8	10	21 - 30
	C	33.7	10	11 - 20
	D	8.7	10	1 - 10

Table 3: Results of the Fischer's LSD Test

The blocks are significantly different if the letter group to which it belongs is different. From Table 3 it may be clearly seen that the first 10 hours with a mean of 8.7 persons in the queue is significantly different from the rest of the groups and is represented by the letter F. The second hour the mean is significantly different from the other groups and is represented by the letter E. Hours 20 to 30 with a mean of 38.8 is not significantly different from the mean queue length for hours 30 to 40 but is different from all other groups. These results justify our earlier conclusion using the theory of runs that steady state is reached after 28 hours. The LSD shows that steady state lies somewhere between hours 20 an 40.

2.3.4 <u>Inferences</u>

This implies that the results of the theory of runs method agrees with the blocking method and can be used as a basis for determining when steady state begins. From these calculations it may be concluded that the Theory of runs using the Wald-Wolfowitz total number of runs test and the length of the longest run is a fairly good measure for determining when steady state begins in the system. The application of the methods to the non-terminating steady state problem results in a close measure of when steady state begins. We can now implement the methods into GPSS/H using HELP blocks.

We should note here that the theory of runs tests are really tests of independence. We have assumed that the data are independent. Using blocks eliminates some of the effects of autocorrelation. A further extension of this work would be to investigate the test for autocorrelation of the observed data under various experimental conditions. Another possible line of investigation is to increase the total number of data points in the test sample to eliminate autocorrelation completely.

2.4.0 Implementation into GPSS/H

On the basis of the results obtained implementation of the FORTRAN subroutine into the GPSS/H programming language has been carried out. The implementation of the program into GPSS/H depends on the use of the HELP block. The program as implemented into GPSS/H is shown in Appendix 6, 7, 8, 9. The program FTINIT has to be called the first time the a FORTRAN subroutine is called into the GPSS/H program. A fairly accurate documentation of the use of a help block is described in the next section in view of the fact that the description on the use of the Help block is almost inaccurate in the GPSS/H manual. The Program gives us the total number of runs in the problem and the length of the runs. The program goes back to the GPSS/H program as soon as the steady state is reached, i.e., when the total number of runs are greater than or equal to 4 and the length of the longest run is less than or equal to 4. The use of the Help block is described in Appendix 12.

2.4.1 Conclusion

The results of using a single server queue results in the average queue length and the instantaneous queue length showing considerable variablity. Based on this it was decided to vary the RMULT values. The process did not exhibit the properties necessary to make the runs tests for steady state valid. The data collection periods were next varied and the instantaneous queue lengths and the average queue lengths were collected over intervals of 1/2 hour, 1 hour, 2 hours, 3 hours and 10 hours. Plotting the collected instantaneous and the average queue lengths show that the process did not exhibit the properties necessary to make the runs tests for steady state valid. The next step was to change utilization from 95% to lower values such as 90%, 85% and 75%, and the average queue lengths do not show properties necessary to make runs tests for steady state valid. At this stage it was necessary to examine whether the random number generators were generating transations in an exponential fashion. The suspicion that the random number generators were not generating transations in an exponential fashion was confirmed. Testing GPSS/H Version 1.9 showed an exponential distribution. At this stage it was decided to introduce a second queue with service times of 135 and return to GPSS/H version 1.0 because the HELP block is not documented adequately. The average hourly queue lengths of the first queue was measured. The average queue lengths of the first queue using the same random number finally resulted in a queue tending toward steady state preceded by an initial transient. This was tested using the Wald-Wolfowitz total number of runs test and the length of the longest run teshnique. The results obtained show that the two method give us a fairly close indication of when steady state begins. This is confirmed by the use of blocking method of Emshoff and Sisson [1970]. We then use Statistical Tests such as ANOVA and Fisher's LSD to show that the steady state begins somewhere between 20th and 40th hours. Based on the results obtained, the method was implemented into GPSS/H version 1.0. The programs necessary for implementation are given in the Appendices. It appears that the use of GPSS/H Version 1.0 random number generator must be validated for randomness for the RMULT values to be used. This is highlighted by the experiments to test the exponenetial interarrival time and the exponential service times. We must investigate the change in results obtained by using the new version of GPSS/H which appears to be far more promising.

Section III

Appendices

Appendix 1

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                 INSTANTANEOUS QUEUE LENGTH                       *
*                                                                  *
* SINGLE SERVER QUEUEING SYSTEM IID EXPONENTIAL SERVICE TIMES *
* AND EXPONENTIAL INTERARRIVAL TIMES AND CUSTOMERS SERVED IN  *
* FIFO MANNER. THE QUEUE LENGTH STATISTICS ARE COLLECTED TO   *
* DETERMINE WHEN THE STEADY STATE BEGINS. THIS PROGRAM IS     *
* USED TO DETERMINE INSTANTANEOUS QUEUE LENGTHS TO DETERMINE  *
* IF STEADY STATE IS ATTAINED FOR INSTANTANEOUS QUEUE LENGTHS *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

       SIMULATE


* INCREASE MEMORY SPACE
       REALLOCATE COM,40000

* TWO RANDOM NUMBER GENERATORS USED FOR FUNCTION 1 EXP1
* AND EXP2. THESE NUMBERS CAN BE CHANGED TO GET DIFFERENT
* RANDOM NUMBER STREAMS.
       RMULT      43127,23891
*
       OPERCOL    25

1      MATRIX     MX,200,1   DEFINES MATRIX  SAVEVALUE
*                            DIMENSIONS


* * * * * * * * * * * * * * * * * * * * * * * * * *
*         EXPONENTIAL INTER-ARRIVAL TIME             *
* * * * * * * * * * * * * * * * * * * * * * * * * *

EXP1   FUNCTION   RN1,C24

0,0/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69

0.6,.915/.7,1.2/.75,1.38/.8,1.6/.84,1.83/.88,2.12

0.9,2.3/.92,2.52/.94,2.81/.95,2.99/.96,3.2/.97,3.5

0.98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7/.9997,8
```

```
* * * * * * * * * * * * * * * * * * * * * * * * * *
*          EXPONENTIAL SERVICE TIME                *
* * * * * * * * * * * * * * * * * * * * * * * * * *

EXP2   FUNCTION   RN2,C24

0,0/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69

0.6,.915/.7,1.2/.75,1.38/.8,1.6/.84,1.83/.88,2.12

0.9,2.3/.92,2.52/.94,2.81/.95,2.99/.96,3.2/.97,3.5

0.98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7/.9997,8

* * * * * * * * * * * * * * * * * * * * * * * * * *
* CREATE CUSTOMERS WITH EXPONENTIAL INTER-ARRIVAL *
* ARRIVAL TIME EXPONENTIAL SERVICE TIME AND THEY  *
* WAIT IN A QUEUES. AFTER THIS THEY GET THE SERV- *
* ICE THEY HAVE COME IN FOR.                      *
* * * * * * * * * * * * * * * * * * * * * * * * * *

      GENERATE   200,FN$EXP1  CREATE A CUSTOMER WITH
*                               EXPONENTIAL INTERARRIVAL TIME

       QUEUE       1   ENTER QUEUE 1

       SEIZE       1   GET HOLD OF THE STOREKEEPER

       DEPART      1   LEAVE QUEUE 1

       ADVANCE     195,FN$EXP2 EXPONENTIAL SERVICE TIME
*                              TAKEN TO SERVICE TIME

       RELEASE     1  RELEASE SHOPKEEPER

       TERMINATE

* * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                 DATA COLLECTION                       *
*                                                       *
* THE QUEUE STATISTICS OF THE FIRST QUEUE ARE COLLECTED *
* FOR THE INSTANTEONEUS QUEUE LENGTHS FOR THE QUEUE AND *
* THESE VALUES ARE SAVED IN THE MATRIX SAVEVALUE        *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * *

       GENERATE   3600    CREATE 3600 SECONDS
* WHEN 3600 IS CHANGED TO 1800, THEN STATISTICS IS COLLECTED
* HALF HOURLY. VARYING A PARAMETER OF THE GENERATE BLOCK IS
* USED TO DETERMINE 1/2 HOURLY, HOURLY, TWO HOURLY INSTANTA-
* NEOUS QUEUE LENGTHS.
```

```
        MSAVEVALUE 1,(201-TG1),1,Q1,MX SAVE THE INSTANTENEOUS
*                                       QUEUE LENGTHS
        TERMINATE  1
*
*
*
        START      200    RUN FOR 200 HOURS

        END
```

Appendix 2

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                    AVERAGE QUEUE LENGTH                        *
*                                                                *
* SINGLE SERVER QUEUEING SYSTEM IID EXPONENTIAL SERVICE TIMES *
* AND EXPONENTIAL INTERARRIVAL TIMES AND CUSTOMERS SERVED IN   *
* FIFO MANNER. THE AVERAGE QUEUE LENGTH IS COLLECTED TO          *
* DETERMINE WHEN THE STEADY STATE BEGINS.                        *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

       SIMULATE

* INCREASE MEMORY SPACE AVAILABLE.
       REALLOCATE COM,40000

* TWO RANDOM NUMBER GENERATORS USED FOR FUNCTION  EXP1
* AND EXP2. BY CHANGING THE RMULT VALUES DIFFERENT RANDOM
* NUMBER STREAMS CAN BE GENERATED.
       RMULT      43127,23891
*
       OPERCOL    25

* * * * * * * * * * * * * * * * * * * * * * * * * * *
*       EXPONENTIAL INTER-ARRIVAL TIME              *
* * * * * * * * * * * * * * * * * * * * * * * * * * *

EXP1   FUNCTION   RN1,C24

0,0/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69

0.6,.915/.7,1.2/.75,1.38/.8,1.6/.84,1.83/.88,2.12

0.9,2.3/.92,2.52/.94,2.81/.95,2.99/.96,3.2/.97,3.5

0.98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7/.9997,8

* * * * * * * * * * * * * * * * * * * * * * * * * * *
*           EXPONENTIAL SERVICE TIME                *
* * * * * * * * * * * * * * * * * * * * * * * * * * *

EXP2   FUNCTION   RN2,C24

0,0/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69

0.6,.915/.7,1.2/.75,1.38/.8,1.6/.84,1.83/.88,2.12

0.9,2.3/.92,2.52/.94,2.81/.95,2.99/.96,3.2/.97,3.5

0.98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7/.9997,8
```

```
* * * * * * * * * * * * * * * * * * * * * * * * *
* CREATE CUSTOMERS WITH EXPONENTIAL INTER-ARRIVAL *
* ARRIVAL TIME EXPONENTIAL SERVICE TIME AND THEY  *
* WAIT IN A QUEUE TO BE SERVICED.                *
* * * * * * * * * * * * * * * * * * * * * * * * *

     GENERATE   200,FN$EXP1  CREATE A CUSTOMER WITH
*                            EXPONENTIAL INTERARRIVAL TIME

     SAVEVALUE  3+,X1*(C1-X2),XF PUT CURRENT QUEUE LENGTH
*   MULTIPLIED BY THE TIME THE QUEUE LENGTH REMAINS THAT VALUE

     SAVEVALUE  2,C1,XF  SAVE CURRENT CLOCK TIME

     SAVEVALUE  1+,1,XF  INCREASE QUEUE LENGTH BY 1

     SEIZE      1        GET HOLD OF THE STOREKEEPER

     SAVEVALUE  3+,X1*(C1-X2),XF AS SOON AS THE STORE
* KEEPER IS SEIZED THE QUEUE LENGTH CHANGES

     SAVEVALUE  2,C1,XF  SAVE CURRENT CLOCK TIME.

     SAVEVALUE  1-,1,XF  DECREASE QUEUE LENGTH BY 1

     ADVANCE    195,FN$EXP2 EXPONENTIAL SERVICE TIME
*                           TAKEN TO SERVICE TIME

     RELEASE    1  RELEASE SHOPKEEPER

     TERMINATE

* * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                  DATA COLLECTION                        *
* THE QUEUE STATISTICS OF THE FIRST QUEUE ARE COLLECTED   *
* AND THE AVERAGE QUEUE LENGTHS FOR EACH HOUR ARE COMPU-  *
* TED AND THIS IS USED TO GET THE AVERAGE QUEUE LENGTH    *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * *

     GENERATE   3600    CREATE 3600 SECONDS
* BY CHANGING THE TIME INTERVAL CREATED THE AVERAGE NUMBER
* OF CUSTOMERS CAN BE VARIED AND THIS CAN BE USED TO CHANGE
* DATA COLLECTION PERIOD.

     SAVEVALUE  4,XF3/3600,XF STORE AVERAGE QUEUE LENGTH

     PRINT      1,4,XF   PRINT THE HOURLY AVERAGE QUEUE LENGTH
```

```
      SAVEVALUE  3,0,XF   SET QUEUE LENGTH TO 0
      TERMINATE  1
*
*
      START      200    RUN FOR 200 HOURS
      END
```

Appendix 3

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*   IS THE INTERARRIVAL TIME AND SERVICE TIME EXPONENTIAL?      *
*                                                               *
* SINGLE SERVER QUEUEING SYSTEM IID EXPONENTIAL SERVICE TIMES *
* AND EXPONENTIAL INTERARRIVAL TIMES AND CUSTOMERS SERVED IN  *
* FIFO MANNER. THE QUEUE LENGTH STATISTICS ARE COLLECTED TO   *
* DETERMINE WHEN THE STEADY STATE BEGINS.                     *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

      SIMULATE

* REALLOCATE MORE MEMORY SPACE
      REALLOCATE COM,40000

* TWO RANDOM NUMBER GENERATORS USED FOR FUNCTION 1 EXP1 AND EXP2
      RMULT       43127,23891
*
      OPERCOL     25

1     MATRIX      MX,200,1   DEFINES MATRIX DIMENSIONS

1     TABLE       IA,0,40,35  IA = INTERARRIVAL

2     TABLE       MP1PF,0,40,35 SERVICE TIME

* * * * * * * * * * * * * * * * * * * * * * * * * * *
*         EXPONENTIAL INTER-ARRIVAL TIME            *
* * * * * * * * * * * * * * * * * * * * * * * * * * *

EXP1   FUNCTION   RN1,C24

0,0/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69

0.6,.915/.7,1.2/.75,1.38/.8,1.6/.84,1.83/.88,2.12

0.9,2.3/.92,2.52/.94,2.81/.95,2.99/.96,3.2/.97,3.5

0.98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7/.9997,8
```

```
* * * * * * * * * * * * * * * * * * * * * * * * * *
*          EXPONENTIAL SERVICE TIME                *
* * * * * * * * * * * * * * * * * * * * * * * * * *

EXP2   FUNCTION   RN2,C24

0,0/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69

0.6,.915/.7,1.2/.75,1.38/.8,1.6/.84,1.83/.88,2.12

0.9,2.3/.92,2.52/.94,2.81/.95,2.99/.96,3.2/.97,3.5

0.98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7/.9997,8


* * * * * * * * * * * * * * * * * * * * * * * * * *
* CREATE CUSTOMERS WITH EXPONENTIAL INTER-ARRIVAL *
* ARRIVAL TIME EXPONENTIAL SERVICE TIME AND THEY  *
* WAIT IN A QUEUE. WE KEEP TRACK OF THE NUMBER OF *
* CUSTOMER ENTERING QUEUE AND THE SERVICE TIMES   *
* * * * * * * * * * * * * * * * * * * * * * * * * *

       GENERATE   200,FN$EXP1,,,,1PF CREATE A CUSTOMER WITH
*                             EXPONENTIAL INTERARRIVAL TIME

       TABULATE   1   TABULATE INTERARRIVAL TIMES

       QUEUE      1   ENTER QUEUE 1

       SEIZE      1   GET HOLD OF THE STOREKEEPER

       DEPART     1   LEAVE QUEUE 1

       MARK       1PF MARK TRANSACTION ENTERS ADVANCE BLOCK

       ADVANCE    195,FN$EXP2 EXPONENTIAL SERVICE TIME
*                             TAKEN TO SERVICE TIME

       RELEASE    1  RELEASE STOREKEEPER

       TABULATE   2  TABULATE SERVICE TIME

       TERMINATE
```

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                DATA COLLECTION                           *
*                                                          *
* THE QUEUE STATISTICS OF THE FIRST QUEUE ARE COLLECTED    *
* FOR THE INSTANTEONEOUS QUEUE LENGTHS FOR THE QUEUE AND   *
* THESE VALUES ARE SAVED IN THE MATRIX SAVEVALUE           *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

      GENERATE   3600    CREATE 3600 SECONDS


      MSAVEVALUE 1,(201-TG1),1,Q1,MX SAVE THE INSTANTENEOUS
*                                        QUEUE LENGTHS

      TERMINATE  1
*
*
*
      START      200    RUN FOR 200 HOURS

      END
```

Appendix 4

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                        USING TWO QUEUES                          *
*                                                                  *
* SINGLE SERVER QUEUEING SYSTEM IID EXPONENTIAL SERVICE TIMES *
* AND EXPONENTIAL INTERARRIVAL TIMES AND CUSTOMERS SERVED IN  *
* FIFO MANNER. THE QUEUE LENGTH STATISTICS ARE COLLECTED TO   *
* DETERMINE WHEN THE STEADY STATE BEGINS. SINCE STEADY STATE  *
* COULD NOT BE ACHIEVED BY USING ONE QUEUE THE CUSTOMER IS    *
* REQUIRED TO GO THROUGH TWO QUEUES TO GET STEADY STATE.      *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

      SIMULATE


* REALLOCATE MORE MEMORY SPACE
      REALLOCATE COM,40000

* TWO RANDOM NUMBER GENERATORS USED FOR FUNCTION 1 EXP1
* AND EXP2. THE RMULT NUMBERS CAN BE CHANGED TO GET
* A DIFFERENT RANDOM NUMBER STREAM.
      RMULT      43127,23891
*
      OPERCOL    25

* * * * * * * * * * * * * * * * * * * * * * * * * * *
*       EXPONENTIAL INTER-ARRIVAL TIME              *
* * * * * * * * * * * * * * * * * * * * * * * * * * *

EXP1   FUNCTION   RN1,C24

0,0/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69

0.6,.915/.7,1.2/.75,1.38/.8,1.6/.84,1.83/.88,2.12

0.9,2.3/.92,2.52/.94,2.81/.95,2.99/.96,3.2/.97,3.5

0.98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7/.9997,8

* * * * * * * * * * * * * * * * * * * * * * * * * * *
*          EXPONENTIAL SERVICE TIME                 *
* * * * * * * * * * * * * * * * * * * * * * * * * * *

EXP2   FUNCTION   RN2,C24

0,0/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69

0.6,.915/.7,1.2/.75,1.38/.8,1.6/.84,1.83/.88,2.12

0.9,2.3/.92,2.52/.94,2.81/.95,2.99/.96,3.2/.97,3.5

0.98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7/.9997,8
```

```
* * * * * * * * * * * * * * * * * * * * * * * * * *
* CREATE CUSTOMERS WITH EXPONENTIAL INTER-ARRIVAL *
* ARRIVAL TIME EXPONENTIAL SERVICE TIME AND THEY  *
* GET TWO SERVICES FOR EACH OF WHICH THE HAVE TO  *
* WAIT IN TWO QUEUES. THIS DONE TO GET THE QUEUE  *
* TO ATTAIN STEADY STATE AFTER A FEW HOURS        *
* * * * * * * * * * * * * * * * * * * * * * * * * *

      GENERATE   200,FN$EXP1  CREATE A CUSTOMER WITH
*                             EXPONENTIAL INTERARRIVAL TIME

      SAVEVALUE  3+,X1*(C1-X2),XF PUT CURRENT QUEUE LENGTH
*   MULTIPLIED BY THE TIME THE QUEUE LENGTH REMAINS THAT VALUE

      SAVEVALUE  2,C1,XF  SAVE CURRENT CLOCK TIME

      SAVEVALUE  1+,1,XF  INCREASE QUEUE LENGTH BY 1

      SEIZE      1        GET HOLD OF THE STOREKEEPER

      SAVEVALUE  3+,X1*(C1-X2),XF AS SOON AS THE STORE
* KEEPER IS SEIZED THE QUEUE LENGTH CHANGES

      SAVEVALUE  2,C1,XF  SAVE CURRENT CLOCK TIME.

      SAVEVALUE  1-,1,XF  DECREASE QUEUE LENGTH BY 1

      ADVANCE    135,FN$EXP2 EXPONENTIAL SERVICE TIME
*                            TAKEN TO SERVICE TIME

      RELEASE    1  RELEASE SHOPKEEPER

      QUEUE      2  ENTER THE 2ND QUEUE

      SEIZE      2  GET SHOPKEEPER2 WHEN HE'S FREE

      DEPART     2  LEAVE THE SECOND QUEUE

      ADVANCE    135,FN$EXP2  EXPONENTIAL PROCESSING TIME

      RELEASE    2  RELEASE SHOPKEEPER2

      TERMINATE
```

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                DATA COLLECTION                           *
*                                                          *
* THE QUEUE STATISTICS OF THE FIRST QUEUE ARE COLLECTED    *
* AND PASSED INTO THE FORTRAN SUBROUTINE FOR CALCULATING   *
* THE RUN LENGTHS ABOUT THE MEAN AND THE TOTAL NUMBER OF   *
* RUNS. THIS IS USED TO DETERMINE WHEN STEADY STATE BEGINS*
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

      GENERATE   3600     CREATE 3600 SECONDS


      SAVEVALUE  4,XF3/3600,XF STORE AVERAGE QUEUE LENGTH

      PRINT      1,4,XF    PRINT THE HOURLY AVERAGE QUEUE LENGTH

      SAVEVALUE  3,0,XF    SET QUEUE LENGTH TO 0

      TERMINATE  1
*
*
*
      START      200     RUN FOR 200 HOURS

      END
```

Appendix 5

```
C************************************************************
C* THIS IS THE FORTRAN PROGRAM USED TO CALCULATE THE TOTAL *
C* NUMBER OF RUNS ABOVE AND BELOW THE MEAN. THE LENGTH OF  *
C* EACH RUN IS ALSO CALCULATED. THESE RESULTS ARE USED TO  *
C* VERIFY OUR HYOTHESIS THE WHEN THE DISTRIBUTION IS COMPL-*
C* RANDOM, I.E., WHEN THE TOTAL NUMBER OF RUNS EXCEED 4 OR *
C* WHEN THE LENGTH OF THE LONGEST RUN IS 4 OR LESS.        *
C************************************************************
$JOB

C$OPTIONS       PAGES=50,LINES=60

      REAL A(300),MEAN(300)

      INTEGER PTCT(10),RUNCT(300),H,G

      I=1

C* READ ALL THE AVERAGE QUEUE LENGTH VALUES OBTAINED AND
C* STORE IT IN AN ARRAY.
      READ(5,*) A(I)

C* A NEGATIVE STOPPING CRITERIA HAS BEEN INSERTED AT THE
C* BOTTOM WHEN THE END OF FILE IS REACHED.
      WHILE (A(I).GE.0) DO

      I=I+1

      READ(5,*) A(I)

      ENDWHILE

      N=I-1

      G=N-9

      DO 10 I=1,G,1

         SUM = 0

         H = I + 9

         DO 20 J =I,H,1

C* ADDING 10 ELEMENTS INTO THE ARRAY
         SUM = SUM + A(J)

 20   CONTINUE
```

```
C* CALCULATING THE MOVING AVERAGE
      MEAN(I) = SUM/10

 10   CONTINUE

      WRITE(6,*)'HOURS   MEAN   TOTAL RUNS
                                   INDIVIDUAL RUN LENGTH'

      DO 30 I = 1,G,1

         K = 1

         KSUM = K

         PTCT(K) = 1

         RUNCT(I) = 1

         H = I + 8

         DO 40 J = I,H,1

C* COMPARING THE VARIOUS VALUES IN THE ARRAY TO DETERMINE
C* THE TOTAL NUMBER OF THE ABOVE RUNS AND BELOW THE MEAN
C* AND  THE RUN LENGTHS OF THE RUNS ABOUT THE MEAN
            IF (((A(J).GT.MEAN(I)).AND.(A(J+1).GT.MEAN(I))).OR.

     *        ((A(J).LT.MEAN(I)).AND.(A(J+1).LT.MEAN(I))).OR.

     *        ((A(J).EQ.MEAN(I)).AND.(A(J+1).EQ.MEAN(I))))

C* INCREASE THE LENGTH OF THE CURRENT RUN BY 1
              THEN  PTCT(K) = PTCT(K) + 1

            ELSE

C* INCREASE THE TOTAL NUMBER OF RUNS BY 1
              RUNCT(I) = RUNCT(I) + 1

              K = K + 1

              KSUM = K

              PTCT(K) = 1

            ENDIF

 40   CONTINUE
```

```
C* WRITE THE MOVING AVERAGE, TOTAL NUMBER OF RUNS AND
C* LENGTH OF EACH RUN
             WRITE(6,21) I, I + 9, MEAN(I), RUNCT(I),
                        (PTCT(M),M=1,KSUM)

 21          FORMAT (I3,I3,F12.6,I3,2X,9(I3,1X))

 30     CONTINUE

       STOP

       END

C* DATA ENTRY AFTER THIS
$ENTRY
4
6
7
8
11
10
12
14
16
18
-1
```

Appendix 6

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*          IMPLEMENTATION INTO GPSS USING HELP BLOCK            *
*                                                               *
* TWO SINGLE SERVER QUEUES ARE USED. THE LENGTH OF THE FIRST   *
* QUEUE IS USED FOR OUR CALCULATIONS. THE GPSS PROGRAM DETER   *
*  -MINES  WHEN STEADY STATE BEGINS AND THE  FORTRAN  ROUTINE  *
*              IS CALLED USING THE HELP BLOCK.                  *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

      SIMULATE

* LOAD STATEMENT USED SINCE THE SUBROUTINE IS CALLED
* 200 TIMES SO AS TO REDUCE COST OF EXECUTION
      LOAD       RUN2

* REALLOCATE MORE MEMORY SPACE
      REALLOCATE COM,40000

* TWO RANDOM NUMBER GENERATORS USED FOR FUNCTION 1 EXP1 AND EXP2
      RMULT      43127,23891
*
      OPERCOL    25

* * * * * * * * * * * * * * * * * * * * * * * * * *
*        EXPONENTIAL INTER-ARRIVAL TIME            *
* * * * * * * * * * * * * * * * * * * * * * * * * *

EXP1  FUNCTION   RN1,C24

0,0/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69

0.6,.915/.7,1.2/.75,1.38/.8,1.6/.84,1.83/.88,2.12

0.9,2.3/.92,2.52/.94,2.81/.95,2.99/.96,3.2/.97,3.5

0.98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7/.9997,8

* * * * * * * * * * * * * * * * * * * * * * * * * *
*          EXPONENTIAL SERVICE TIME                *
* * * * * * * * * * * * * * * * * * * * * * * * * *

EXP2  FUNCTION   RN2,C24

0,0/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69

0.6,.915/.7,1.2/.75,1.38/.8,1.6/.84,1.83/.88,2.12

0.9,2.3/.92,2.52/.94,2.81/.95,2.99/.96,3.2/.97,3.5

0.98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7/.9997,8
```

```
* * * * * * * * * * * * * * * * * * * * * * * * * * *
* CREATE CUSTOMERS WITH EXPONENTIAL INTER-ARRIVAL *
* ARRIVAL TIME EXPONENTIAL SERVICE TIME AND THEY  *
* GET TWO SERVICES FOR EACH OF WHICH THE HAVE TO  *
* WAIT IN TWO QUEUES. THIS DONE TO GET THE QUEUE  *
* TO ATTAIN STEADY STATE AFTER A FEW HOURS        *
* * * * * * * * * * * * * * * * * * * * * * * * * * *

      GENERATE   200,FN$EXP1  CREATE A CUSTOMER WITH
*                             EXPONENTIAL INTERARRIVAL TIME

      SAVEVALUE  3+,X1*(C1-X2),XF PUT CURRENT QUEUE LENGTH
*   MULTIPLIED BY THE TIME THE QUEUE LENGTH REMAINS THAT VALUE

      SAVEVALUE  2,C1,XF  SAVE CURRENT CLOCK TIME

      SAVEVALUE  1+,1,XF  INCREASE QUEUE LENGTH BY 1

      SEIZE      1        GET HOLD OF THE STOREKEEPER

      SAVEVALUE  3+,X1*(C1-X2),XF AS SOON AS THE STORE
*            KEEPER IS SEIZED THE QUEUE LENGTH CHANGES

      SAVEVALUE  2,C1,XF  SAVE CURRENT CLOCK TIME.

      SAVEVALUE  1-,1,XF  DECREASE QUEUE LENGTH BY 1

      ADVANCE    135,FN$EXP2 EXPONENTIAL SERVICE TIME
*                            TAKEN TO SERVICE TIME

      RELEASE    1  RELEASE SHOPKEEPER

      QUEUE      2  ENTER THE 2ND QUEUE

      SEIZE      2  GET SHOPKEEPER2 WHEN HE'S FREE

      DEPART     2  LEAVE THE SECOND QUEUE

      ADVANCE    135,FN$EXP2  EXPONENTIAL PROCESSING TIME

      RELEASE    2  RELEASE SHOPKEEPER2

      TERMINATE

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                 DATA COLLECTION                          *
* THE QUEUE STATISTICS OF THE FIRST QUEUE ARE COLLECTED    *
* AND PASSED INTO THE FORTRAN SUBROUTINE FOR CALCULATING   *
* THE RUN LENGTHS ABOUT THE MEAN AND THE TOTAL NUMBER OF   *
* RUNS. THIS IS USED TO DETERMINE WHEN STEADY STATE BEGINS*
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

Appendix 6 (contd.)

```
      GENERATE   3600    CREATE 3600 SECONDS

      SAVEVALUE  4,XF3/3600,XF STORE AVERAGE QUEUE LENGTH

      PRINT      1,4,XF   PRINT THE HOURLY AVERAGE QUEUE LENGTH

      HELPB      RUN2,4   CALL FORTRAN SUBROUTINE TO CALCULATE
* THE MOVING AVERAGE OF THE QUEUE LENGTH, TOTAL NUMBER OF RUNS
* AND THE LENGTH OF EACH RUN

      SAVEVALUE  3,0,XF   SET QUEUE LENGTH TO 0

      TERMINATE  1
*
*
*
      START      200    RUN FOR 200 HOURS

      END
```

Appendix 7

```
C* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C*  SUBROUTINE TO CALCULATE THE TOTAL NUMBER OF RUNS    *
C*  AND LENGTH OF EACH RUN ABOUT THE MEAN. IF THE        *
C*  TOTAL NUMBER OF RUNS IS 4 OR MORE THEN THE SYSTEM   *
C*  IS SAID  TO BE IN UNSTEADY STATE. THE LENGTH OF     *
C*  THE LONGEST RUN IF GREATER THAN 4 IS ALSO USED      *
C*  USED AS A CRITERIA FOR THE UNSTEADY STATE.          *
C*  THIS SUBROUTINE IS CALLED 200 TIMES BY GPSS         *
C* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

      SUBROUTINE RUN2(IX)
C* PARAMETER PASSED IS SAVEVALUE XF4 FROM THE MAIN PROGRAM

      REAL MEAN(200)

      INTEGER A(200),PTCT(10),RUNCT(200),H,G

C* INTEGER I INITIALIZED TO 1
      INTEGER I/1/

C* THIS SUBROUTINE IN ASSEMBLY LANGUAGE HAS TO BE THE
C* THE FIRST EXECUTABLE STATEMENT IN ANY FORTRAN CODE
C* THIS CODE IS IN THE FILE FTINIT ASSEMBLE A AND THE
C* COMPILED VERSION IS IN THE FILE FTINIT TEXT A
C* FTINIT IS CALLED ONLY THE FIRST TIMETHOUGH THE
C* SUBROUTINE IS CALLED 200 TIMES BY THE GPSS PROGRAM
      CALL FTINIT

C* THE SAVEVALUES PASSED INTO THE SUBROUTINE IS STORED
C* IN THE ARRAY A(I)
      A(I)=IX

      I = I + 1

C* IF THE VALUE OF I IS LESS THEN 200 THEN RETURN
C* TO THE MAIN PROGRAM
      IF (I.LT.200) RETURN

C* THESE STATEMENTS WILL BE EXECUTED ONLY IF 200 VALUES
C* ARE IN THE ARRAY
      N = 199

      G = N - 9

      DO 10 I=1,G,1

         SUM = 0

         H = I + 9
```

```
        DO 20 J = I,H,1

C* ADDING  10 ELEMENTS IN THE ARRAY

        SUM = SUM + A(J)

 20   CONTINUE

C* CALCULATING THE MOVING AVERAGE
      MEAN(I) = SUM/10

 10   CONTINUE


      DO 30 I = 1,G,1

        K = 1

        KSUM = K

        PTCT(K) = 1

        RUNCT(I) = 1

        H = I + 8

        DO 40 J = I,H,1

C* COMPARING THE VARIOUS VALUES IN THE ARRAY TO DETERMINE
C* TOTAL NUMBER OF RUNS ABOVE AND BELOW THE MEAN AND THE
C* RUN LENGTHS OF THE VARIOUS RUNS ABOUT THE MEAN
          IF (((A(J).GT.MEAN(I)).AND.(A(J+1).GT.MEAN(I))).OR.

     *        ((A(J).LT.MEAN(I)).AND.(A(J+1).LT.MEAN(I))).OR.

     *        ((A(J).EQ.MEAN(I)).AND.(A(J+1).EQ.MEAN(I)))) THEN

C* INCREASE THE LENGTH OF THE CURRENT RUN BY 1
             PTCT(K) = PTCT(K) + 1

          ELSE

C* INCREASE THE TOTAL NUMBER OF RUNS BY 1
            RUNCT(I) = RUNCT(I) + 1

            K = K + 1

            KSUM = K

            PTCT(K) = 1
          ENDIF
 40    CONTINUE

        IF (PTCT(K).LT.4).AND.(RUNCT(I).GE.4)
        WRITE(6,21) I+9
 21     FORMAT (IX,'STEADY STATE BEGINS AFTER',I5,'HOURS')
        RETURN
           ELSE
        WRITE(6,*)'NO STEADY STATE EXISTS IN THE PROBLEM'
 30     CONTINUE
C* RETURN TO THE GPSS PROGRAM
       RETURN

       END
```

(This is the assembly language code called FTINIT ASSEMBLE A that has to be complied and CALL FTINIT must be the first executable statement in the FORTRAN subroutine. The file called FTINIT TEXT A is created when the program called FTINIT ASSEMBLE A is compiled.)

```
FTINIT   CSECT
         USING FTINIT,15
         CLI   NOTFIRST,0
         BNER  14
         MVI   NOTFIRST,1
         STM   13,14,SAVE1314
         L     13,4(0,13)
         L     15,=V(VSCOM#)
         BAL   14,64(0,15)
         USING *,14
         LM    13,14,SAVE1314
         BR    14
SAVE1314 DS    2A
NOTFIRST DC    X'00'
         END
```

Appendix 9

(This is the JCL necessary for running the GPSS program with a HELP block calling an external FORTRAN subroutine. The GPSS/H program is called H2 GPSS A and the FORTRAN subroutine is called RUN2 FORTRAN A. You may change the GPSS/H program name and the FORTRAN subroutine name depending on the names used by you. They would have to accordingly changed in the JCL.)

```
//*++ PRINT VMMSG SERVICE * TIME 0,10 EXPAND
// EXEC FORTVCL
//SYSIN DD *
//*$$ INCLUDE RUN2 FORTRAN
//LKED.SYSLMOD DD DSN=&&TEMPLIB(RUN2),DISP=(NEW,PASS),
//   SPACE=(CYL,(1,1,1)),UNIT=SYSDA
// EXEC GPSSH
//STEPLIB DD
//        DD DSN=SYS1.VFORTLIB,DISP=SHR
//HELPLIB DD DSN=&&TEMPLIB,DISP=(OLD,KEEP)
//SYSIN DD *
//*$$ INCLUDE H2 GPSS
```

Appendix 10

Quantiles of the Wald-Wolfowitz Total Number of Runs and Length of Longest run for 10 data points

This is the part of the table taken from Practical Nonparametric Statistics and total number of runs required and the maximum length of the longest run permitted for a completely random distribution is calculated according to the formula:

$$W_p = \frac{2mn}{m+n} + 1 + x_p \sqrt{\frac{2mn(2mn-m-n)}{(m+n)^2\ (m+n+1)}}$$

The formula is used to fill out points combinations not covered by the table, and the value of x_p is the standard p quantile of a standard normal random variable. The m's and the n's represent the aboves and the belows about the mean respectively.

m	n	Total Number of runs($w_{.05}$)	Length of longest run($w_{.05}$)
2	8	4	4
3	7	4	4
4	6	4	4
5	5	4	4

[Conover, p. 414, 1980]

Appendix 11

On Using the Help block

The use of the help block in GPSS is a fairly complicated procedure and the attempt here is to describe the use as simply as possible. There are basically three types of Help blocks used in Fortran. They are HELPA, HELPB and HELPC.

The key factors to remember in using any Fortran subroutine is that:

1. The fortran subroutine name should be the same as the file name.

2. The first excutable statement in the Fortran subroutine should be CALL FTINIT.

3. A program called FTINIT ASSEMBLE should exist in the memory along with a compiled version called FTINIT TEXT. This is shown in Appendix 9.

4. The Fortran subroutine need not be compiled as indicated in the manual if the JCL shown in Appendix 9 is used but must be compiled for running the program in CMS. It is recommended that using CMS should be the preferred alternative when using external Help routines. In the program shown it may be necessary to type in the command

```
filedef 6 disk fn ft
```

where the filename and the file type are different from the output of GPSS. Otherwise the output is produced on the screen. Care should be taken to not name the filename and filetype the same as the output file of the GPSS program as this may cause problems. It must also be remembered that the JCL must be contained in a seperate file and to run the GPSS file the JCL file must be run. It may also be noted that the JCL file we include RUN2 FORTRAN the H2 GPSS which are my file-names. These may be replaced by the filenames used by the person running the GPSS program with a FORTRAN include file.

5. It is not possible to pass values from one Fortran subroutine to the other by using COMMON/BLK1. It may be necessary to call a subroutine as many times as necessary. The cost of calling a program in terms of computer time can be reduced by having a LOAD statement immediately subsequent to the SIMULATE statement.

6. Any errors in the FORTRAN subroutine complicates the process of running the GPSS program and extreme care must be taken to ensure that the FORTRAN program to be included is running as required independently.

Section IV

References

References

Bobblier, P.A.,Kahan, B.C., and Probst, A.R., (1976), Simulation with GPSS and GPSS V, New Jersey: Prentice Hall.

Conover, W.J., 1980, Practical Non-Parametric Statistics, New York: Johm Wiley & Sons, pp. 340 - 371.

Conway, R.W., 1963, Some Tactical Problems in Digital Simulation, Vol. 10, No. 1, pp. 47 - 61.

Conway, R.W., Johnson, B.M., Maxwell, W.L., 1959, Some Problems of Digital System Simulation, 5th International Convention of Institute of Management Sciences, pp. 92 - 110.

Cooper, Robert B., 1977, Introduction to Queueing theory, New York: North Holland, pp. 71 - 89.

Emshoff, James R. and Sisson, Roger L., 1970, Design and use of Computer Simulation Models, London: MacMillan Company, pp. 189 - 196.

Fishman, G.S., 1973, Statistical Analysis of Queueing Simulation, Management Science, Vol. 20, pp. 363 - 369.

Greenberg, S., 1972, GPSS Primer, New York: Wiley InterScience.

Henriksen, J.O., and Crain, R.C., 1984, GPSS/H User's Manual, Wolverine Software Company.

Kleijnen, Jack P.C., 1975, Statistical Techniques in Simulation, Part I, pp. 85 - 200.

Kleijnen, Jack P.C., 1975, Statistical Techniques in Simulation, Part II, pp. 451 - 510.

Kleijnen, Jack P.C., 1988, Analysing Simulation Experiments with common random numbers, Management Science, Vol. 34, No. 1, pp. 65 - 74.

Law, Averill M., and Kelton, W. David, 1982, Simulation Modelling and Analysis, New York: McGraw Hill.

Massey, W.A., 1984, Open Networks of Queues: Their Algebraic Structure and Estimating their transient behavior, Advanced Appled Probability, Vol. 16, 176 - 201.

Pidd, M., 1984, Computer Simulation in Management Science, Chichester: John Wiley & Sons.

Schmidt, J.W., and Taylor, R.E., 1970, Simulation and Analysis of Industrial Management Systems, Homewood, Illinois: Richard D. Irwin, Inc., pp. 59 - 91.

Sharma, S.N., 1965, Theory of Runs, Kansas State University, pp. 1 - 20.

Thran, M.K., Cumulative Rank Sum Test: Theory and Applications, Kansas State University, pp. 1 - 13A.

Wadsworth, Harrison M., Stephens, Kenneth S., Godfrey, Blanton A., 1984, Modern Methods for Quality Control, New York: John Wiley & Sons.

USING RUN TESTS TO ELIMINATE INITIAL UNSTEADY STATE DURING SIMULATION

by

H.GOPALKRISHNA MENON

B.Sc (Met. Engg.), Sambalpur University, 1984

AN ABSTRACT OF A REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

INDUSTRIAL ENGINEERING

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1988

There are three aspects to digital simulation of a problem once it has been identified and the model formulated, which include the model implementation, strategic planning and tactical planning to determine how the test runs specified in the experimental design are to be implemented. One of the tactical planning problems is starting the experiment avoiding the artificial bias introduced by the starting conditions.

The literature currently available on the subject suggests three techniques to overcome the initial transient:

1. Exclude data from some initial period from consideration.
2. Initial conditions to make transient as short as possible.
3. Long runs to make the starting conditions insignificant.

These result in bias to the results or we may have to use long runs to eliminate the effects of initial transient.

For this, after simulating a problem with initial transient state starting conditions, runs tests will be used to determine when steady state is reached. The problem of single server queue problem with 95% utilization rate was used. Considerable difficulties were faced and the problem had to be modified to get an average queue length reaching steady state after a few hours. The methods used are maximum run length technique and the distribution of run lengths. We then verify using CUSUM charts to determine the accuracy of the methods. Finally, if the methods are found acceptable then it may be incorporated into GPSS to automatically eliminate the initial transient.