Sampling Variability Effects in Input Modeling in Discrete-Event Simulation

Rob McGregor

CS 420 UMSA

April 29, 2002

Discrete-event simulation attempts to give meaningful information about a system's performance by executing a model of that system based upon data collected from the system. Therefore, the reliability of the simulation output is heavily in¤uenced by the reliability of the input data. Simply running additional simulations may not give better system response estimates if the input data is uncertain. So a meaningful question arises as to how much of an impact the sampling variability of the data collected for the input model of a simulation has on the output statistics for the simulation.

To start off, we look at the simple model of a $M/M/1$ queue as shown in Figure 1. The model represents a system in which customers are arriving with exponential time between arrivals at a rate of λ, i.e. the mean interarrival time is $\frac{1}{\lambda}$. The service times are also exponential with a rate of μ and a mean of $\frac{1}{\mu}$. When the traf£c intensity $\rho = \frac{\lambda}{\mu} < 1$, the steady-state expected time in the queue is £nite. Clearly the values of λ and μ are system dependent, so they must be determined by collecting data for the system which is to be simulated. For our tests, we will assume that $\lambda = 1$

and $\mu = \frac{10}{9}$. The traf£c intensity for this queue is $\rho = \frac{9}{10}$.

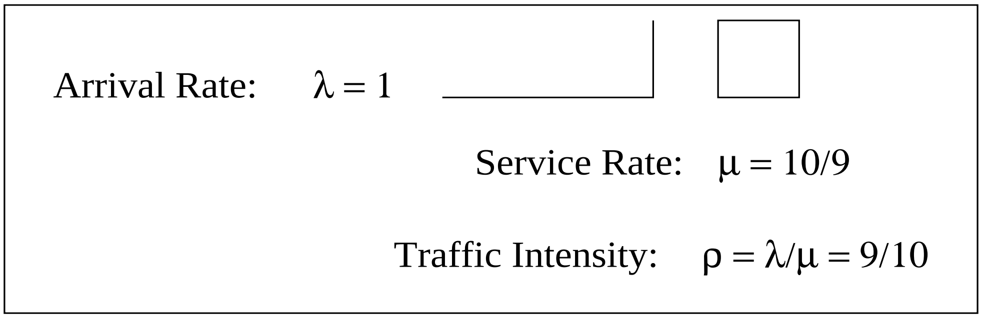

Figure 1. Basic $M/M/1$ queue

In order to determine the effect of sampling variability on output statistics for our $M/M/1$ queue we sample n exponential(λ) interarrival times, $X_1, X_2, \ldots, X_n$. The estimated mean interarrival time from the n sampled times is $\overline{X}$, and $1/\overline{X}$ is the estimated arrival rate. Figure 2 shows the probability density function of $\overline{X}$ for $n = 12$, which is an Erlang(1, 12) distribution. This distribution is centered over the expected value of $\overline{X}$, which is 1.

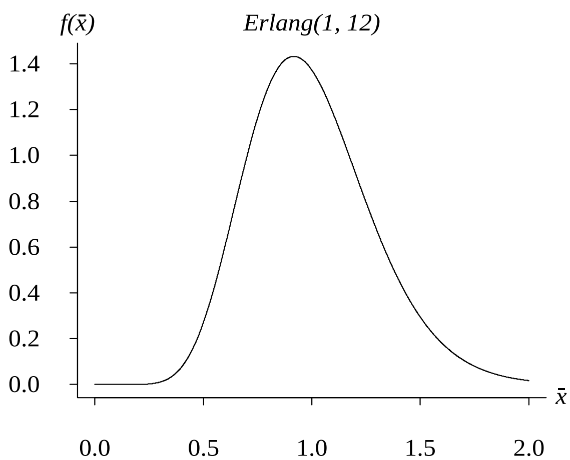

Figure 2. Distribution of $\overline{X}$ when n = 12

Similarly, sample m exponential(μ) service times, $Y_1, Y_2, \ldots, Y_m$. Then our estimated mean service time is $\overline{Y}$ and the estimated service rate is $1/\overline{Y}$. Figure 3 shows the probability density function of $\overline{Y}$ when $m = 10$ which is an Erlang($\frac{10}{9}$, 10) distribution which is centered over the expected value of $\overline{Y}$, 0.9. In general the sample mean of k independent exponential(ν) random variables has an Erlang(ν, k) distribution.

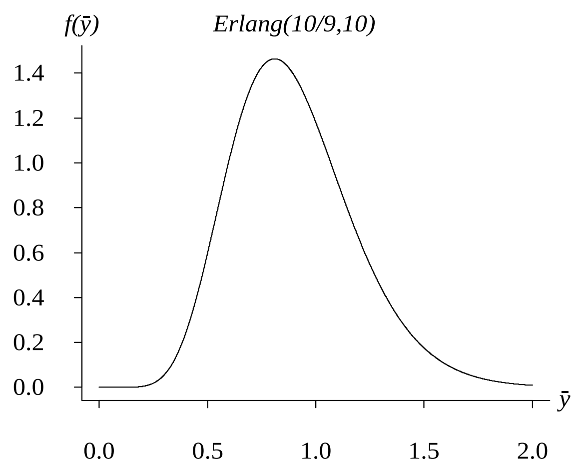

Figure 3. Distribution of $\overline{Y}$ when m = 10

One thing that can occur when using $\overline{X}$ and $\overline{Y}$ as the arrival rate and service rate in the simulation model that could not occur when simulating with the £xed values of λ and μ when $(\lambda < \mu)$ is that we can now have a traf£c intensity greater than 1, which will occur when $\overline{X} < \overline{Y}$. In other words, when the average interarrival rate is greater than the rate of service, the size of the queue will eventually grow without bound since jobs are arriving faster than they are being serviced. It is useful to know what the probability is of this occurring, e.g. $P(\overline{X} < \overline{Y}) = \int_0^\infty \int_{\bar{x}}^\infty f_{\bar{x}}(\bar{x}) f_{\bar{y}}(\bar{y}) d\bar{y} d\bar{x}$. A visualization of this probability is shown in £gure 4. This £gure shows contours of the joint PDF of $\overline{X}$ and $\overline{Y}$ and the line represents when $\overline{X} = \overline{Y}$. So the area under the joint PDF of $\overline{X}$ and $\overline{Y}$ that is above the line is the probability of the traf£c intensity being greater than 1 when $n = 12$ and $m = 10$.

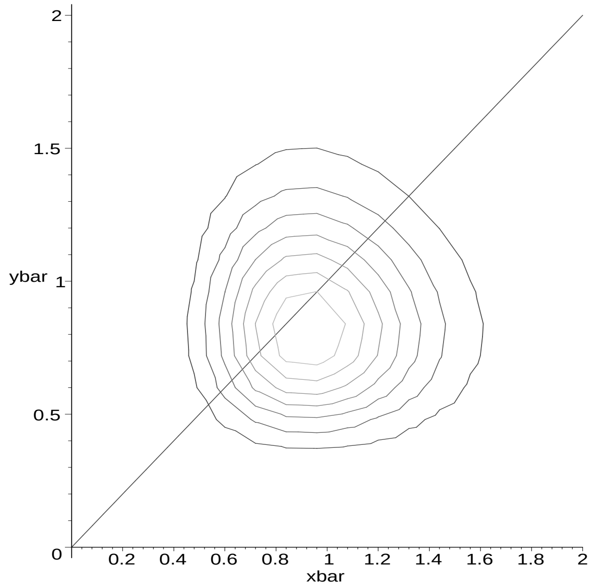

Figure 4. Contours of the joint PDF of the sample mean $\overline{X}$

and the sample mean $\overline{Y}$ when n = 12 and m = 10

In order to see how collecting more service times or interarrival times will affect the probability of the queue eventually growing without bound, the probabilities for values of n and m adjacent to $n = 12$ and $m = 10$ are shown in Figure 5.

		m		
		9	10	11
	11	.4025	.4031	.4035
n	12	.3983	.3987	.3990
	13	.3946	.3949	.3950

Figure 5. Probabilities of unde£ned queue lengths (traf£c intensity $>$ 1) for different m and n values.

It may seem counter-intuitive that in Figure 5 when m is larger, meaning more samples of service times with a mean of 0.9 were taken, that the probability of a traf£c intensity being larger than 1 is increasing. However, as shown in Figure 6, eventually as m increases this probability will start to decrease, especially for higher values of n. As expected, the probabilities decrease as n decreases.

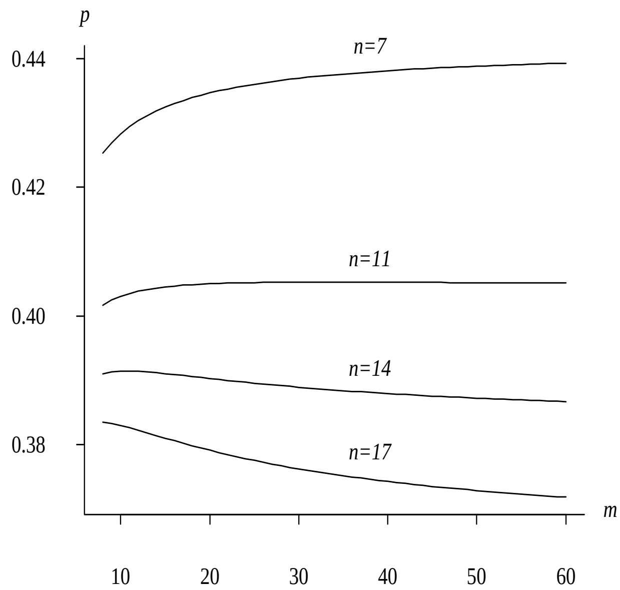

Figure 6. Probabilities of $\rho > 1$ for changing m and n values.

The distribution of the delay times is another output statistic that can be considered. We start by looking at the delay time of the third customer. Let D_3 be the expected delay of the third customer since this will be de£ned for all values of ρ. Kelton (1985) has computed the expected delay times of the k^{th} customer for a standard $M/M/1$ queue with £xed λ and μ values. Using Kelton's formula, $E[D_3]$, the expected delay of the third customer is

$$\frac{\lambda}{\mu^2}\left[\frac{1+4(\frac{\lambda}{\mu})+2(\frac{\lambda}{\mu})^2}{(\frac{\lambda}{\mu}+1)^3}\right].$$

For the values chosen, $\lambda = 1$ and $\mu = \frac{10}{9}$, using the equation above, $E[D_3] = 0.7345$. As a double check on my implementation of the $M/M/1$ queue as well as a check on Kelton's formula and my derivation of the third delay time from this formula, the $M/M/1$ queue was simulated

5,000,000 times with an average delay of the 3^{rd} customer being 0.7344, con£rming the validity of the equation and the implementation. In addition to testing for these values of λ and μ, it is also to necessary to check the validity of Kelton's formula for values such that $\rho = 1$ and $\rho > 1$. This is because when we are using estimated parameters, we have seen that it is not guaranteed that $\rho < 1$. To check for $\rho = 1$, λ and μ were both set to 1, and from the formula, $E[D_3]$ = 0.875. The simulation con£rmed the result. For $\rho > 1$, $\lambda = 1$ was used and $\mu = \frac{1}{1.1}$. The formula gave $E[D_3]$ = 1.022, which was also con£rmed exactly by the simulation.

In order to now £nd the distribution of the delay times, use Kelton's formula which we now assume to be true for all values of ρ. In place of λ in the equation, we use the distribution of $1/\overline{X}$ and in place of μ in the equation, we use the distribution of $1/\overline{Y}$. Since APPL is designed to be able to form distributions using operations on other distributions, it seems like the right thing to use to £nd this distribution. However, the problem proved to be too computationally intensive to be solved using APPL, so an analytical distribution of the third delay has not been determined. Even when attempting to compute the distribution of the second delay time, which uses a far simpler equation, APPL was unable to compute it.

The next step is to simulate the third delay time with the estimated parameters $1/\overline{X}$ and $1/\overline{Y}$ rather than the £xed values for λ and μ. The values for $\overline{X}$ and $\overline{Y}$ are computed for $n = 12$ and $m = 10$ on each replication, and then the simulation is run using $\overline{X}$ and $\overline{Y}$, and the delay of the third customer is calculated. The distribution of these delay times as compared to the distribution of the delay times for the constant parameters is shown in Figure 7. For 1,000,000 replications, the average delay of the third customer for the estimated parameters was 0.7853 as compared to the average delay with the £xed parameters being 0.7345, which is about a 6.9% increase. So while the average delay for the estimated parameters is higher, as the distribution shows there are also slightly more zero delay times with the estimated parameters. This can be explained by the fact that the estimated parameters will tend to spread out the delay times compared to the constant parameters.

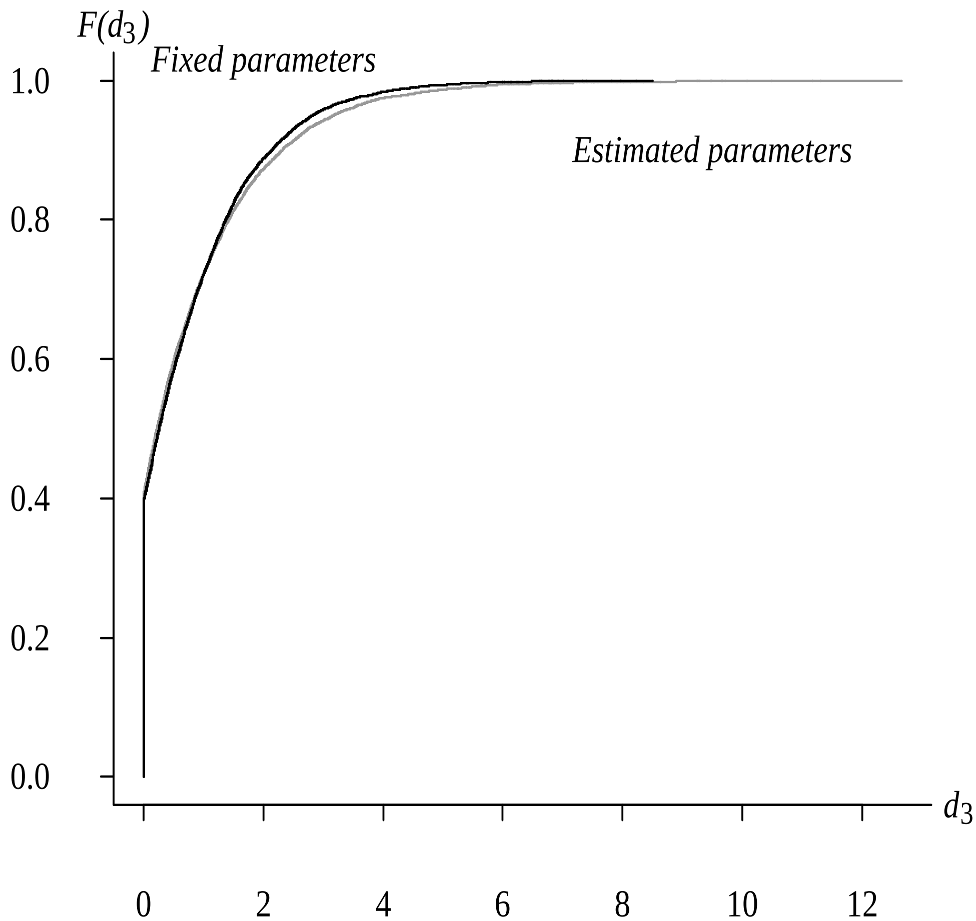

Figure 7. Distribution of delay times for £xed parameters and estimated parameters.

REFERENCES

Kelton, W. David. 1985. The Transient Behavior of the $M/M/s$ Queue, with Implications for Steady-State Simulation. *OperationsResearch*, Vol. 33, No.2, 378-395.

APPENDIX

kthdelay.c: Computes the average delay of the kth item in the queue and output each delay to a data £le.

```
#include <stdio.h>
#include <math.h>
#include "rngs.h"
#include "rvgs.h"

#define LAST         3L                    /* number of jobs processed */
#define START        0.0                   /* initial time             */

   double GetArrival(double arrival)
/* ------------------------------
 * generate the next arrival time
 * ------------------------------
 */
{
  //static double arrival = START;
  SelectStream(0);

  arrival += Exponential(1.0);
  return (arrival);
}


   double GetService(void)
/* ------------------------------
 * generate the next service time
 * ------------------------------
 */
{
  SelectStream(1);
  return (Exponential(0.9));
}


  int main(void)
{
```

```
  FILE *fd;
  int delayed;
  const int K = 3;
  const int loops = 10000;
  double kthdelay = 0.0;
  long index;
  int i;
  double arrival, delay, service, wait, departure;
  struct {                      /* sum of ...           */
      double delay;             /*   delay times        */
      double wait;              /*   wait times         */
      double service;           /*   service times      */
      double interarrival;      /*   interarrival times */
  } sum = {0.0, 0.0, 0.0};
  fd = fopen("delays50.d", "w");
  delayed = 0;

  PlantSeeds(123456789);
  for (i = 0; i < loops; i++){
    index     = 0;                  /* job index            */
    arrival   = START;              /* time of arrival      */
    departure = START;              /* time of departure    */

    while (index < LAST) {
      index++;
      arrival      = GetArrival(arrival);
      if (arrival < departure)
delay      = departure - arrival;  /* delay in queue    */
      else
delay      = 0.0;                  /* no delay          */
      if (index == K)
kthdelay += delay;
      service      = GetService();
      wait         = delay + service;
      departure    = arrival + wait;       /* time of departure */
      sum.delay   += delay;
      sum.wait    += wait;
      sum.service += service;
    }
    sum.interarrival += arrival - START;
    if(delay > 0)
      delayed++;
    fprintf(fd, "%f ", delay);
  } // for
```

```
  printf("   average delay of %dth item = %6.4f\n",
 K, kthdelay / (double(loops)));
  fprintf(fd, "\n");
  return (0);
}
```

kthdelayns.c: Computes delay times for the kth item using an estimated arrival rate with n = 12 and an estimated service rate with m = 10

```
#include <stdio.h>
#include <math.h>
#include "rngs.h"
#include "rvgs.h"

#define LAST         3L          /* number of jobs processed */
#define START        0.0         /* initial time             */

   double GetArrival(double arrival, double aTime)
/* ------------------------------
 * generate the next arrival time
 * ------------------------------
 */
{
  //static double arrival = START;
  SelectStream(0);

  arrival += Exponential(aTime);
  return (arrival);
}


   double GetService(double sTime)
/* ------------------------------
 * generate the next service time
 * ------------------------------
 */
{
  SelectStream(1);
  return (Exponential(sTime));
}

double findRate(double averageTime, int num)
     /*
       computes an average time based on num random variables
```

```
       with a mean of averageTime
     */
{
  double sum = 0.0;         // sum of times
  SelectStream(3);
  for (int i = 0; i < num; i++)
    sum += (Exponential(averageTime));
  return (double) (sum / double (num));
} // findRate

  int main(void)
{
  int delayed;
  FILE *fd;
  const int K = 3;
  const int loops = 10000;
  double kthdelay = 0.0;
  long index;
  double aTime, sTime;      // average arrival and service times
  double arrival, delay, service, wait, departure;
  struct {                             /* sum of ...           */
      double delay;                    /*   delay times        */
      double wait;                     /*   wait times         */
      double service;                  /*   service times      */
      double interarrival;             /*   interarrival times */
  } sum = {0.0, 0.0, 0.0};

  fd = fopen("delays50n.d", "w");
  delayed = 0;

  PlantSeeds(123456789);
  for (int i = 0; i < loops; i++){
    index     = 0;                      /* job index            */
    aTime = findRate(1.0, 12);
    sTime = findRate(0.9, 10);
    arrival   = START;                  /* time of arrival      */
    departure = START;                  /* time of departure    */
    while (index < LAST) {
      index++;
      arrival      = GetArrival(arrival, aTime);
      if (arrival < departure)
delay      = departure - arrival;   /* delay in queue    */
      else
delay      = 0.0;                   /* no delay          */
      if (index == K)
```

```
kthdelay += delay;
      service      = GetService(sTime);
      wait         = delay + service;
      departure    = arrival + wait;        /* time of departure */
      sum.delay   += delay;
      sum.wait    += wait;
      sum.service += service;
    }
    sum.interarrival += arrival - START;
    if (delay > 0)
      delayed += 1;
    fprintf(fd, "%f ", delay);
  } // for

  printf("   average delay of %dth item = %6.4f\n",
         K, kthdelay / (double(loops)));
  fprintf(fd, "\n");
  return (0);
}
```

Splus5 £le to produce Figure 7 from the data £les produced by the 2 C programs.

```
y1 <- scan("delays50.d")
y2 <- scan("delays50n.d")
y1 <- sort(y1)
y2 <- sort(y2)
n <- length(y1)
longdelay <- max(y1,y2)
# x <- seq(0,ceiling(longdelay), by = 0.01)
postscript(file = "delays50.ps", width = 3.8, height = 3.1, horizon-
tal = F)
par(mai = c(0.5, 0.5, 0.3, 0.3))
plot(c(0,0), c(0,0), xlab="", ylab="",
   sub="", cex = 0.6,
   bty = "l", las = c(1),
   xlim = c(0, longdelay), ylim = c(0, 1),
   lty = c(1), col = c(1), type = "l",
   font = 3)
for (j in 1: n){
segments(y2[j],(j-1)/n, y2[j+1], (j-1)/n, col=2)
segments(y2[j+1], (j-1)/n, y2[j+1], j/n, col=2)
}
for (i in 1: n){
segments(y1[i],(i-1)/n, y1[i+1], (i-1)/n)
```

```
segments(y1[i+1],(i-1)/n,y1[i+1], i/n)
}
text(2.0, 1.03, "Fixed parameters", cex = 0.6, font = 10)
text(9.5, 0.9, "Estimated parameters", cex = 0.6, font = 10)
text(13.5, -0.04, "d", cex = 0.6, font = 10)
text(13.75, -0.052, "3", cex = 0.5, font = 3)
text(-0.5, 1.08, "F(d  )", cex = 0.6, font = 10)
text(-0.25, 1.068, "3", cex = 0.5, font = 3)
dev.off()  # this will shut down the postscript device
```

Maple Commands to produce the Figure 4, the contours.

```
restart;
lambda := 1;
mu := 10/9;
n := 12;
m := 10;
with(plots);
c :=
contourplot({
lambda^n*n^n*xbar^(n-1)*(1/(n-1)!)*exp(-n*lambda*xbar)*
mu^m*m^m*ybar^(m-1)*(1/(m-1)!)*exp(-m*mu*ybar)},xbar=0..3,
ybar=0..3);
x := plot(x -> x, 0..2);
display(c, x);
```

Maple Commands to produce delay time data

```
restart;
read('APPL');
lambda := 1;
mu := 10/9;
fd := fopen("integrands.d", WRITE);
for n from 11 by 3 to 17 do
for m from 8 by 1 to 60 do
fprintf(fd, "%f ", evalf(int(int((lambda^n)*(n^n)*xbar^(n-1)*(exp(-
n*lambda*xbar))*(1/(n-1)!)*(mu^m)*(m^m)*(ybar^(m-1))*(exp(-m*mu*ybar))*(1/(
1)!),ybar =xbar..infinity),xbar=0..infinity)));
end do;
fprintf(fd, "\n");
end do;
```

Splus5 input £le to produce Figure 6 from data £le of delay times

```
y <- matrix(scan("integrands.d"), nrow=53, ncol=4)
x <- c(8:60)
postscript(file = "integrands.ps", width = 3.8, height = 3.1, hor-
izontal = F)
par(mai = c(0.5, 0.5, 0.3, 0.3))
matplot(x, y, xlab="", ylab="",
   sub="", cex = 0.6,
   bty = "l", las = c(1),
   xlim = c(min(x), max(x)), ylim = c(min(y), max(y)),
   lty = c(1), col = c(1), type = "l",
   font = 3)
text(65, 0.37, "m", cex = 0.6, font = 10)
text(6, 0.447, "p", cex = 0.6, font = 10)
text(38, 0.379, "n=17", cex = 0.6, font = 10)
text(38, 0.392, "n=14", cex = 0.6, font = 10)
text(38, 0.409, "n=11", cex = 0.6, font = 10)
text(38, 0.442, "n=7", cex = 0.6, font = 10)

   dev.off()  # this will shut down the postscript device
```