Midterm Exam for Simulation (CAP 4800) Summer 2013

>>> SOLUTIONS <<<

Welcome to the Midterm Exam for *Simulation*. Read each problem carefully. There are 10 required problems where each problem is worth 10 points. There is also an additional extra credit problem worth 10 points. You may have with you a calculator, pencils and/or pens, erasers, blank paper, and one 8.5 x 11 inch "formula sheet". On this formula sheet you may have anything you want (definitions, formulas, homework answers, old exam answers, etc.) as **handwritten by you in pencil or ink** on both sides of the sheet. Photocopies, scans, or computer generated and/or printed text are not allowed on this sheet. Note to tablet (iPad, etc.) users – you may **not** print-out your handwritten text for the formula sheet. Please answer the problems on the available sheets. If you need extra space, use the back side of the page of the problem that you are answering. You have 120 minutes for this exam.

Problem #1

Sub-problems (a) thru (b) worth 2 points each, (c) and (d) worth 3 points each.

Answer the following questions regarding the basics of modeling.

a) Give the catalog description of this course. In other words, what is this course all about?

This course is an introduction to discrete-event simulation for performance modeling of computer and communication systems. At the completion of this course, a student will be able to model a system and predict its performance.

b) What is a system (give a more formal definition that "whatever your draw a box around)?

A set of interacting or interdependent entities forming an integrated whole

c) What is a model? Give a short formal definition.

"A model is a representation (physical, logical, or functional) that mimics another object under study." (Molloy 1989).

d) What is a simulation? Give a short formal definition.

"Computer simulation is the discipline of designing a model of an actual or theoretical physical system, executing the model on a computer, and analyzing the execution output." (Fishwick, 1995

Problem #2 Sub-problems (a) thru (c) worth 2 points each, (d) worth 4 points.

Answer the following questions regarding performance and design of experiments.

a) Give one performance metric of primary interest to a user of a system and give one performance metric of primary interest to the owner or operator of a system.

Users care about delay (want it as low as possible), operators care about utilization (want it as high as possible)

b) Fill in the blanks in the following sentence,

"Experiments are a method of investigating _causal_ relationships among _variables_."

c) Define response variable, factor, factor level, and replication.

Response variable = performance measure of interest (the outcome of the experiment)
Factor = variable that affect the response variable
Factor level = values that a factor can take on
Replication = the repetition or trial of an experiment

d) Describe full factorial and simple experimental design. What the advantages and disadvantages of each?

Full factorial design = every possible combination of factors and factor levels. Simple design = vary one factor at a time (for all its levels). Factorial has advantage of measuring all factor interactions and disadvantage of large number of experiments. Simple is the exact opposite (interactions may be missed, but small number of experiments).

Problem #3 Sub-problems (a) thru (c) worth 2 points each, (d) worth 4 points.

Answer the following questions regarding probability theory.

a) Why do we (in this class studying performance modeling of computer and communication systems) care about probability theory?

Most events in computer and communication systems are characterized by randomness.

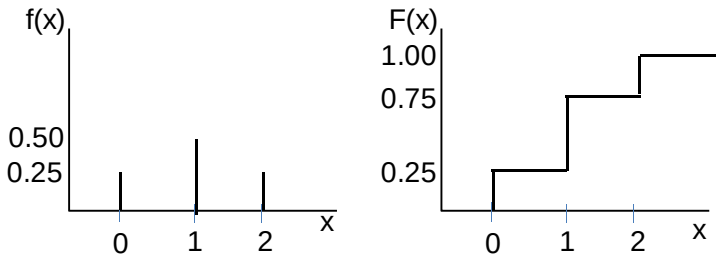
b) Give the experimental definition of probability.

$$\Pr[\text{outcome}] = \lim_{n\to\infty} \frac{\text{Number of observed outcomes}}{n \text{ repetitions of experiment}}$$

c) Define random variable.

A random variable is a function that maps a real number to every possible outcome in the sample space.

d) Plot the pdf and CDF for the random variable X where X is the number of heads that appear when two fair coins are flipped.



Problem #4 Sub-problems (a) and (b) worth 3 points each, (c) worth 4 points.

Answer the following questions regarding probability theory.

a) Assume that the interarrival times of requests to a web server follow an exponential distribution and assume that the rate of arrivals is 2 request per second. What is the probability that the next request will arrive between 1 second and 3 seconds from the time of arrival of the previous request? Show your work. The pdf and CDF for an exponential distribution are:

$$f(t) = \lambda e^{-\lambda t} \quad t \geq 0$$

$$F(t) = 1 - e^{-\lambda t} \quad t \geq 0$$

$$Pr[\text{next arrival between 1 and 3 seconds}] = F(3) - F(1) = \left(1 - e^{-2\cdot 3}\right) - \left(1 - e^{-2\cdot 1}\right) = 0.133$$

b) For the same assumption as (a). What is the probability that the next request will arrive after 3 seconds from the time of arrival of the previous request? Show your work.

$$Pr[\text{next arrival occurs after 3 seconds}] = 1 - F(3) = 1 - \left(1 - e^{-2\cdot 3}\right) = e^{-6} = 0.00248$$

c) The time between arrivals in a Poisson distribution are exponentially distributed. Show (or derive) this. The pdf for a Poisson distribution is:

$$f(k) = \frac{(\lambda t)^k}{k!} e^{-\lambda t} \quad k = 0, 1, 2, \ldots \text{ and } t \geq 0$$

Given than $F(t) = \Pr[T \leq t] = 1 - \Pr[T > t]$ and that $\Pr[t > T]$ is the probability of no events occurring in $[0, T) = \Pr_o[t]$ we have, $F(t) = 1 - \Pr_0[t]$, $\Pr_0[t] = \frac{(\lambda t)^0}{0!} e^{-\lambda t} = e^{-\lambda t}$ which is then $F(t) = 1 - e^{-\lambda t}$ and this is the CDF of an exponential distribution.

Problem #5 Sub-problems (a) worth 2 points, (b) and (d) worth 4 points each

Answer the following questions regarding random number generation.

a) Fill in the blank in the following sentence

A sequence of 0s and 1s can be considered to be random if it is _not_ possible to _predict_ the next bit in the sequence with better than _0.5_ probability.

b) Describe two ways to test for independence (or lack thereof) for a uniform(0, 1) RNG.

Autocorrelation can be measured for many lags and should be zero for all lags. A 2D plot of a sequence of pairs (x1, x2) can be used to visually detect patterns.

c) Will an RNG implemented using a linear congruential generator generate independent values for all lags? Explain your answer.

No because at some point it will complete its cycle and repeat values. The cycle may be many billions of values, but sooner or later it will repeat. Thus, it cannot be independent for all lags.

Problem #6 Each sub-problem worth 5 points.

Answer the following questions regarding workload. For both questions you may assume that you have a function `randUnif()` that returns a uniformly distributed random value between 0 and 1 as a double (the function does not need to be seeded).

a) Given the pdf as follows (call this distribution "my distribution"):

$$f(x) = \begin{cases} 2 \cdot x & 0 \le x \le 1 \\ 0 & x > 1 \end{cases}$$

Write a C function that returns a random value distributed as $f(x)$. **Hint:** You may need to convert from $f(x)$ to $F(x)$.

First we convert to F(x) so that we can invert…

$$F(x) = \int_0^x f(y)dy = \int_0^x (2 \cdot y)dy = y^2 \Big|_0^x = x^2$$

Then we can invert F(x) as follows…

$$U = x^2$$

$$x = \sqrt{U}$$

The C function is then:

```
double myDistribution()
{
  double z;          // Unif(0, 1) random value

  // Pull a unif(0, 1) random value
  z = randUnif();

  // Compute my distribution random value using inversion
  value = sqrt(z);

  return(value);
}
```

b) Assume you have made measurements on the size of embedded images in web pages. You have measured the size of 1 million images and found that 500,000 images were exactly 10 KBytes, 250,000 images were exactly 20 Kbytes, 100,000 images were exactly 30 KBytes, and 150,000 images were exactly 50 KBytes. Image sizes are independent of each other. Write a C function that returns an image size based on an empirical distribution from the measurements made.

```
int genMyDistribuiton()
{
  double z;
  int    size;

  z = rand_val(0);
  if (z <= (0.50) size = 8 * 10 * 1024;
  else if (z <= (0.50 + 0.25)) size = 8 * 20 * 1024;
  else if (z <= (0.50 + 0.25 + 0.10)) size = 8 * 30 * 1024;
  else size = 8 * 50 * 1024;

  return(size);
}
```

Problem #7 Sub-problems (a) and (b) worth 2 points each, (c) and (d) worth 3 points each.

Answer the following questions about queueing.

a) What is Kendall notation. Describe it.

A queue is described as A/S/c/k/m where A is the arrival distribution, S is the service distribution, c is the number of servers, k is the capacity of the system in number of customers, and m is the number of customers in the universe. A and S can be "M" for Markov, "D" for deterministic, "G" for general, and others.

b) State Little's Law.

$L = \lambda W$ for L is the mean number in the system, λ is the arrival rate, and W is the mean wait in the system

c) Given an M/M/1 queue with an arrival rate of 5 jobs per second and a service rate of 10 jobs per second what is the mean number of customers in the system? What is the mean customer delay?

For an M/M/1 queue we know that $L = \frac{\rho}{1-\rho}$. We know that $\rho = \frac{\lambda}{\mu} = \frac{5}{10} = 0.50$. Solving for $L = \frac{0.50}{1-0.50} = 1.0$ customers. Using Little's Law we have $W = L/\lambda = 1.0/5 = 0.20$ seconds.

d) Repeat (c) for an M/D/1 queue.

Here we need to use the P-K formula and we know that $C_s = 0$ for deterministic service. The P-K formula is,

$$L = \rho + \frac{\rho^2\left(1 + C_s^2\right)}{2(1-\rho)}$$

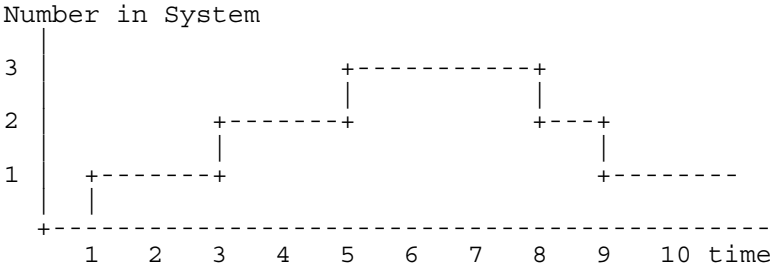
So $L = 0.50 + \frac{0.50^2(1+0)}{2(1-0.50)} = 0.75$ customers. Using Little's Law we have $W = L/\lambda = 0.75/5 = 0.15$ seconds.

Problem #8 Graph is 3 pts, formulas are 3 pts, and numerical solutions are 4 pts.

Consider the following single-server queueing system from time $t = 0$ to $t = 10$ sec. Arrivals and service times are:

- Customer #1 arrives at $t = 1$ and requires 3 seconds of service time
- Customer #2 arrives at $t = 3$ and requires 4 seconds of service time
- Customer #3 arrives at $t = 4$ and requires 1 second of service time
- Customer #4 arrives at $t = 5$ and requests 3 seconds of service time

Solve for system throughput (X), total busy time (B), mean service time (Ts), utilization, (U), mean system time (W), and mean number in the system (L). You must show your work to receive credit!



X = C/T = 3/10 = 0.3 cust/sec
B = 9 seconds
Ts = B/C = 9/3 = 3 sec/cust
U = B/T = 9/10 = 0.90 or 90%
W = Sum of w / C = 18 / 3 = 6 sec
L = Sum of w / T = 18 / 10 = 1.8 customers.

Problem #9 Each bug identified and fixed is 1 point

In the appendix to this exam is our `mm1.c` simulation program for modeling an M/M/1 queue with some bugs. The bugs are all major in that they will affect the execution and output (but, the program still compiles). Identify the bugs and fix them. You can do your work in the appendix.

See appendix for solution.

Problem #10 Sub-problems (a) and (b) worth 3 points each, (c) is worth 4 points.

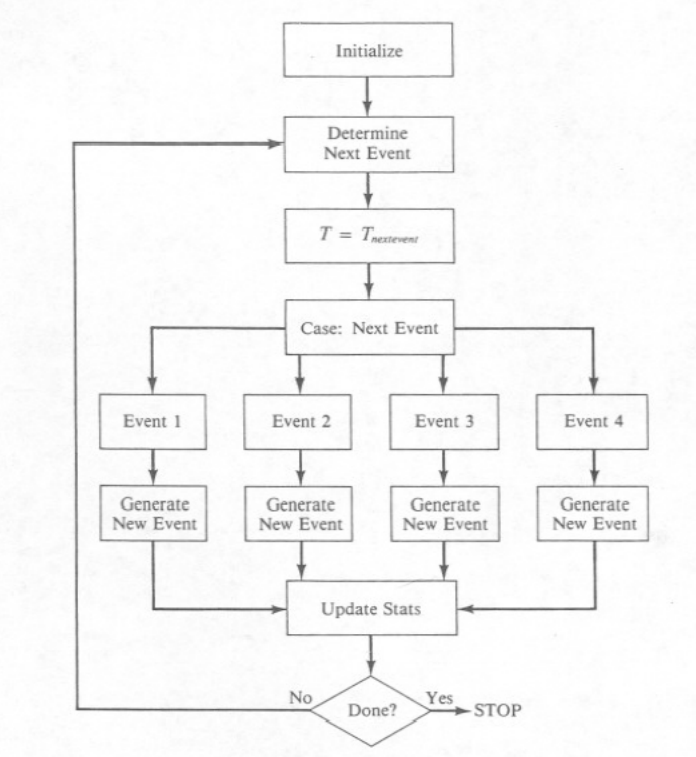
a) What are the components of a discrete event simulation? **Hint:** system state is a key component.

The components are system state, simulation clock (or time), event list, event routine, statistical counters, library routines, report generator, initialization, and main program.

b) Explain what an event list is, what it contains, and how it is used by event routines. What type of data structure is used to implement an event list?

The event list in a DES is a time-ordered list of events. Each event minimally contains a time (of occurrence) and id. Event routines change the system state based on the next event in the event list, the event routines then generate the next event, which is inserted into the event list in time-order. Typically event lists are implemented as linked lists.

c) Sketch the flowchart of a generic discrete event simulation.



From Molloy (1989)

Extra Credit

Basic loop structure is 5 points, key line is 3 points, the rest is 2 points.

Write a Monte Carlo simulation to solve for π (or, rather, estimate π). You may assume that you have a function named `randUnif()`that returns a uniformly distributed random value between 0 and 1 as a double (the function does not need to be seeded). Your code does not need to include a header.

```
#include <stdio.h>
#include <math.h>
#define NUM_ITER 10000000
int main()
{
  double   x, y;
  double   len
  double   hitCount;
  double   pi_est;
  int      i;

  hitCount = 0.0;
  for (i=0; i<NUM_ITER; i++)
  {
    x = rand_val(0);
    y = rand_val(0);
    len = sqrt(pow(x, 2.0) + pow(y, 2.0));
    if (len <= 1.0) hitCount++;
  }
  pi_est = 4.0 * (hitCount / NUM_ITER);
  printf("Estimated pi = %f \n", pi_est);

  return(0);
}
```

Humor

Hmm…. maybe studying is not a good idea after all…

No Study = Fail
Study = No Fail
\+ _____

No Study + Study = Fail + No Fail

➔ (~~No +~~ 1) Study = (~~No +~~ 1) Fail

∴ Study = Fail

From: http://totalgadha.com/tgtown/mangoman/2010/03/07/126-and-still-counting/

I hope that everyone did well ☺

Appendix A

```
//============================================================= file = mm1.c =====
//=  A simple "straight C" M/M/1 queue simulation with bugs                      =
//================================================================================
//=  Notes:                                                                      =
//=   1) This program is adapted from Figure 1.6 in Simulating Computer          =
//=      Systems, Techniqyes and Tools by M. H. MacDougall (1987).               =
//=   2) The values of SIM_TIME, ARR_TIME, and SERV_TIME need to be set.         =
//=-----------------------------------------------------------------------------=
//=  Build: gcc mm1.c -lm, bcc32 mm1.c, cl mm1.c                                 =
//=-----------------------------------------------------------------------------=
//=  Execute: mm1                                                                =
//=-----------------------------------------------------------------------------=
//=  History: KJC (03/09/99) - Genesis                                           =
//=           KJC (05/23/09) - Added RNG function (so not to use compiler RNG)   =
//=           KJC (05/24/09) - Added updated of busy time at end of main loop    =
//=           KJC (06/14/13) - Creates bugs for exam #1 summer 2013              =
//================================================================================
//----- Include files ------------------------------------------------------
#include <stdio.h>              // Needed for printf()
#include <stdlib.h>             // Needed for exit() and rand()
#include <math.h>               // Needed for log()

//----- Constants ----------------------------------------------------------
#define SIM_TIME   1.0e6        // Simulation time
#define ARR_TIME   1.25         // Mean time between arrivals
#define SERV_TIME  1.00         // Mean service time

//----- Function prototypes ------------------------------------------------
double rand_val(int seed);      // RNG for unif(0,1)
double exponential(double x);   // Generate exponential RV with mean x

//===== Main program ==========================================================
int main(void)
{
  double   end_time = SIM_TIME; // Total time to simulate
  double   Ta = ARR_TIME;       // Mean time between arrivals
  double   Ts = SERV_TIME;      // Mean service time
  int      time = 0.0;          // Simulation time
  double   t1 = 0.0;            // Time for next event #1 (arrival)
  double   t2 = SIM_TIME;       // Time for next event #2 (departure)
  unsigned int n = 0;           // Number of customers in the system
  unsigned int c = 0;           // Number of service completions
  double   b = 0.0;             // Total busy time
  double   s = 0.0;             // Area of number of customers in system
  double   tn = time;           // Variable for "last event time"
  double   tb;                  // Variable for "last start of busy time"
  double   x;                   // Throughput
  double   u;                   // Utilization
  double   l;                   // Mean number in the system
  double   w;                   // Mean residence time

  // Seed the RNG
  rand_val(1);

  // Main simulation loop
  while (end_time < time)
  {
    if (t1 < t2)                // *** Event #1 (arrival) ***
    {
      time = t1;
```

double

time < end time

```
      s = s + n * (tn - time);  // Update area under "s" curve
      n++;
      tn = time;                // tn = "last event time" for next event
      t1 = time + exponential(Ta);
      if (n == 1)
      {
        tb = time;              // Set "last start of busy time"
        t2 = time + exponential(Ta);
      }
    }
    else if (t1 < t2)          // *** Event #2 (departure) ***
    {
      time = t2;
      s = s + n * (tn - time);  // Update area under "s" curve
      n--;
      tn = time;                // tn = "last event time" for next event
      c++;                      // Increment number of completions
      if (n > 1)
        t2 = time + exponential(Ta);
      else
      {
        t2 = SIM_TIME;
        b = b + time - tb;      // Update busy time sum if empty
      }
    }
  }

  // End of simulation so update busy time sum
  b = b + time - tb;

  // Compute outputs
  x = c / time;   // Compute throughput rate
  u = b / time;   // Compute server utilization
  l = s / time;   // Compute mean number in system
  w = l / x;      // Compute mean residence or system time

  // Output results
  printf("============================================================= \n");
  printf("=           *** Results from M/M/1 simulation ***           = \n");
  printf("============================================================= \n");
  printf("=  Total simulated time         = %3.4f sec   \n", end_time);
  printf("============================================================= \n");
  printf("=  INPUTS:                                    \n");
  printf("=    Mean time between arrivals = %f sec      \n", Ta);
  printf("=    Mean service time          = %f sec      \n", Ts);
  printf("============================================================= \n");
  printf("=  OUTPUTS:                                   \n");
  printf("=    Number of completions      = %ld cust    \n", c);
  printf("=    Throughput rate            = %f cust/sec \n", x);
  printf("=    Server utilization         = %f %%       \n", 100.0 * u);
  printf("=    Mean number in system      = %f cust     \n", l);
  printf("=    Mean residence time        = %f sec      \n", w);
  printf("============================================================= \n");

  return(0);
}
```

time - tn

Ts

elide

time - tn

n--;

0

Ts

```
//=========================================================================
//= Multiplicative LCG for generating uniform(0.0, 1.0) random numbers    =
//=   - x_n = 7^5*x_(n-1)mod(2^31 - 1)                                    =
//=   - With x seeded to 1 the 10000th x value should be 1043618065       =
//=   - From R. Jain, "The Art of Computer Systems Performance Analysis," =
//=     John Wiley & Sons, 1991. (Page 443, Figure 26.2)                  =
//=   - Seed the RNG if seed > 0, return a unif(0,1) if seed == 0         =
//=========================================================================
double rand_val(int seed)
{
  const long  a =      16807;  // Multiplier
  const long  m = 2147483647;  // Modulus
  const long  q =     127773;  // m div a
  const long  r =       2836;  // m mod a
  static long x;               // Random int value (seed is set to 1)
  long        x_div_q;         // x divided by q
  long        x_mod_q;         // x modulo q
  long        x_new;           // New x value

  // Seed the RNG
  if (seed != 0) x = seed;

  // RNG using integer arithmetic
  x_div_q = x / q;
  x_mod_q = x % q;
  x_new = (a * x_mod_q) - (r * x_div_q);
  if (x_new > 0)
    x = x_new;
  else
    x = x_new + m;

  // Return a random value between 0.0 and 1.0
  return((double) x / m);
}

//=========================================================================
//=  Function to generate exponentially distributed RVs using inverse method   =
//=    - Input:  x (mean value of distribution)                           =
//=    - Output: Returns with exponential RV                              =
//=========================================================================
double exponential(double x)
{
  double z;                    // Uniform random number from 0 to 1

  // Pull a uniform RV (0 < z < 1)
  do
  {
    z = rand_val(0);
  }
  while ((z == 0) || (z == 1));

  return(-x * log(z));
}
```