

>>> SOLUTIONS <<<

Welcome to the comprehensive Final Exam for *Simulation*. Read each problem carefully. There are eight required problems (each worth 12 points – you get 4 points for correctly following these instructions). There is also an additional extra credit question 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. You have 130 minutes for this exam. **Please use a separate sheet of paper for the answer to each question.** Good luck and be sure to show your work!

Problem #1

Each sub-problem worth 4 points.

Answer the following questions regarding the basics of modeling.

a) 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).

b) There are four possible ways to study a system, which are 1) do a mathematical analysis, 2) build a simulation model and experiment on it, 3) build a prototype system and experiment on it, and 4) build the actual system and experiment on it. Which is the “best”? How does simulation compare? Hint: first define the criteria for comparison.

First we must define the criteria, which are 1) accuracy (or fidelity), 2) insight, 3) extensibility, 4) run time, and 5) time (or cost) to develop.

- A mathematical model is poor for (1), best for (2), good for (3), (4), and (5).
- A simulation model is good for (1), medium for (2) and (3), poor for (4), and medium for (5).
- A prototype is good for (1), good for (2), poor for (3), (4), and (5).
- A real systems is best for (1), best for (2), poor for (3) and (4), and very poor for (5)

c) Given a model with factors A and B, A takes on levels A1, A2, and A3 and B takes on levels B1 and B2, describe a full-factorial experimental design and a simple experimental design for this model (that is, give factor levels for each experiment needed). What is the total number of experiments for the full-factorial design? For the simple design? What is the disadvantage of simple versus full-factorial design?

The factor level settings for the full factorial design are: (A1 B1), (A2 B1), (A3 B1), (A1 B2), (A2 B2), and (A3 B2).
The total number of experiments needed is $3 \times 2 = 6$ (3 levels for A and 2 levels for B).

One set of possible factor level settings for a simple design are: (A1 B1), (A2 B1), (A3 B1), and (A3 B2). The total number of experiments needed is $1 + (3 - 1) + (2 - 1) = 4$ (3 levels for A and 2 levels for B).

A simple design may miss the effects of key factor interactions (since not all combinations are considered as they are in a full factorial design).

Problem #2 (Appendix A contains formulas for key distributions)

Each sub-problem worth 4 points.

Answer the following questions regarding probability theory (specific to distributions and expectation).

a) What is a probability density function? What is a Probability Distribution Function? What are the key properties of each?

A probability density function $f(x)$ is simply a probability function for a random variable X where

$$f(x) = \Pr[X = x].$$

The properties are $f(x) \geq 0$ and $\sum_{i=0}^{\infty} f(x_i) = 1$.

A Probability Distribution Function $F(x)$ is the sum of the probability density function values,

$$F(x_k) = \sum_{i=0}^k f(x_i) \text{ for discrete and } F(x) = \int_{y=0}^x f(y) dy \text{ for continuous.}$$

The properties are $F(-\infty) = 0$, $F(\infty) = 1$, and $F(x)$ is monotonically increased for x increasing.

b) Derive the expression for mean (μ) for a uniformly distributed, continuous random variable with minimum value a and maximum value b .

$$\mu = \int_a^b xf(x)dx = \int_a^b x \left(\frac{1}{b-a} \right) dx = \frac{a+b}{2}$$

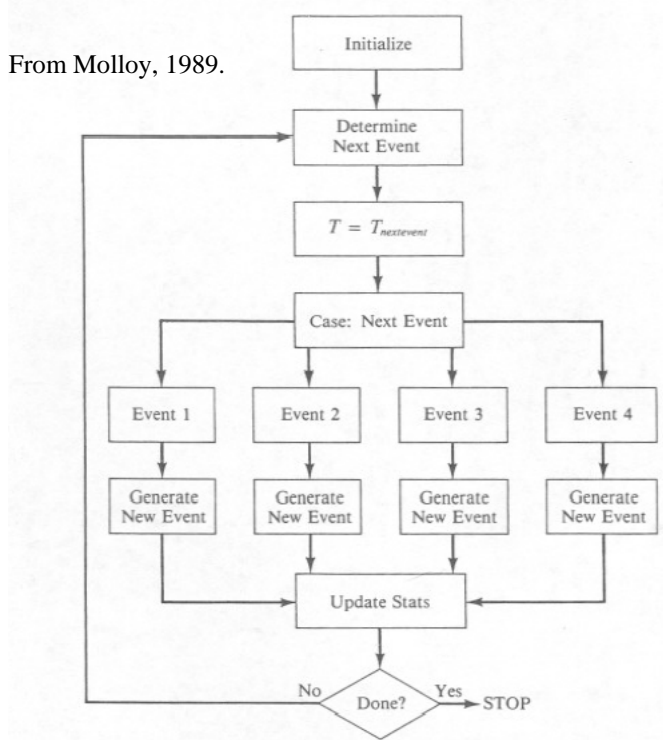
c) The time between arrivals in a Poisson distribution are exponentially distributed. Show (or derive) this.

Given that $F(t) = \Pr[T \leq t] = 1 - \Pr[T > t]$ and that $\Pr[t > T]$ is the probability of no events occurring in $[0, T) = \Pr_0[t]$ 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 #3

Each box worth 1 point (need show only one "Event" and "Generate New Event" sequence). Description of time moving forward is 4 points.

Sketch the flowchart of (or write pseudocode for) a generic discrete event simulation. Explain how time moves forward.



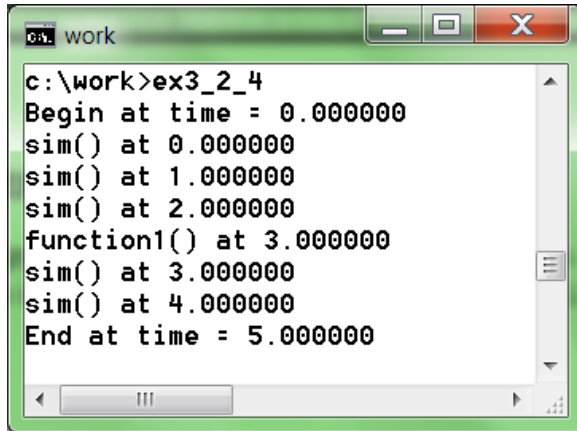
Time moves forward with each event pulled off the event list. An event structure on the list ordered by time minimally contains the event id and time of event. Thus, time will jump forward as determined by events on the event list.

Problem #4

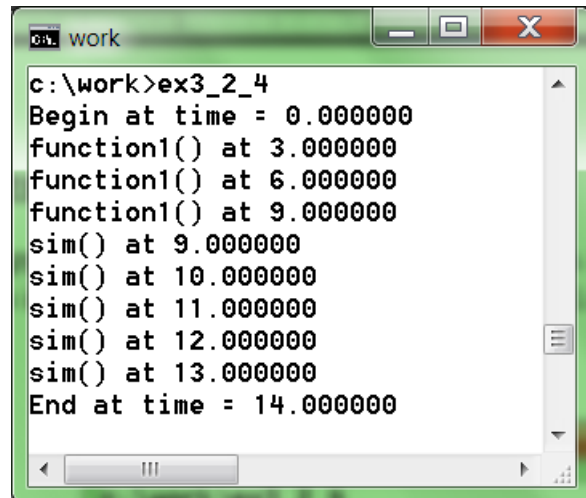
Each case is 6 points. Roughly 1 point per output line. Note that two events that occur at the same time can be shown in any order.

Appendix B contains a simple CSIM program. Give the output from the program for two cases, case 1: program as is, and case 2: the line `create("function1");` in `function1()` removed.

The runtime results for the two cases are:



```
c:\work>ex3_2_4
Begin at time = 0.000000
sim() at 0.000000
sim() at 1.000000
sim() at 2.000000
function1() at 3.000000
sim() at 3.000000
sim() at 4.000000
End at time = 5.000000
```



```
c:\work>ex3_2_4
Begin at time = 0.000000
function1() at 3.000000
function1() at 6.000000
function1() at 9.000000
sim() at 9.000000
sim() at 10.000000
sim() at 11.000000
sim() at 12.000000
sim() at 13.000000
End at time = 14.000000
```

Problem #5

Solution and grading criteria are in Appendix C

Appendix C contains a CSIM model of an M/M/1 queue. Modify this program to collect statistics on arrivals that “see” odd and even number of customers in the system. That is, a customer that arrives to a system with 1, 3, 5, ... customers in the queue plus service sees an odd number of customers. The output should look something like:

```
Arrivals that see an odd number of customers is __ %
Arrivals that see an even number of customers is __%
```

You may make your changes in the appendix (that is, on the exam pages).

Problem #6

Solution and grading criteria are in Appendix D

Appendix D contains a CSIM model of an M/M/1 queue. Modify this program to contain two queues – one with exponential service time with rate μ_1 and the other with deterministic service time with rate μ_2 . Customers that are generated are sent to the first queue and as they depart from this queue they should be sent to the second queue. Customers then leave the system from the second queue. Collect and output statistics on the delay in each queue, utilization of each server, and end-to-end delay (that is time from customer generated to customer departing the second queue). The output should look something like:

```
Delay in queue #1 = __sec
Delay in queue #2 = __ sec
Utilization of server in queue #1 = __ %
Utilization of server in queue #2 = __ %
E2E delay = __ sec
```

You may make your changes in the appendix (that is, on the exam pages).

Problem #7 Solution and grading criteria are in Appendix E

Appendix E contains the disk process for our CSIM model of a disk drive. This model does not check if the size of the requested file is greater than the size of the disk cache. Modify the model so that if the requested file is larger than the cache that a cache hit does NOT occur and that the request is served by the disk platter. Keep statistics on how many requests had file size greater than cache size. You will need to declare some new variables (which could be global and be defined in sim()). Be sure to identify and describe these variables (and don't forget to explain how they should be initialized). You may make your changes in the appendix (that is, on the exam pages).

Problem #8 3 pts for mean, 3 pts for sample std dev, 3 pts for H, and 3pts for interpretation.

You have simulated two systems for 7 replications each. The sample means for response times from system #1 are 110, 105 110, 108, 102, and 112 milliseconds, and from system #2 they are 108, 95, 100, 80, 100, and 110 milliseconds. Can you, with 95% confidence (also 90% confidence), state that system #2 has lower response time than system #1? Why or why not? Show your work. A T-score table is given below.

Selected values of $t_{\alpha/2;N-1}$

	$\alpha/2 = 0.05$	$\alpha/2 = 0.025$
N - 1	t	t
4	2.13	2.78
5	2.02	2.57
6	1.94	2.45
7	1.90	2.37
8	1.86	2.31
9	1.83	2.26
10	1.81	2.23
11	1.80	2.20
12	1.78	2.18

First we determine the difference between sample means (for system #1 minus system #2):

- 0 milliseconds
- 10
- 10
- 28
- 2
- 2

It certainly looks like system #2 is better (i.e., has lower true mean response time).

$$\bar{Y} = \frac{1}{6} \cdot (0 + 2 + 10 + 10 + 28 + 2 + 2) = 9$$

$$S = \sqrt{\frac{1}{6-1} \cdot [(2-9)^2 + (10-9)^2 + \dots + (2-9)^2]} = 10.1$$

$t_{\alpha/2} = 2.57$ for 95% confidence and $t_{\alpha/2} = 2.02$ for 90% confidence (we use $\alpha = 0.05$ and $N = 6$ degrees of freedom, so we use the $N - 1 = 5$ row)

$$H_{95} = 2.57 \cdot \left(\frac{10.1}{\sqrt{6}}\right) = 10.60$$

$$H_{90} = 2.02 \cdot \left(\frac{10.1}{\sqrt{6}}\right) = 8.33$$

So, we can say with 95% confidence that the population mean (true mean) lies between -1.6 and 19.6 (that is, 9 ± 10.6). Since this CI is **not** entirely above zero, we cannot say with 95% confidence that system #2 is better (better = lower response time in this case) than system #1. With 90% confidence the true population mean lies between 0.66 and 17.33 and thus we can say with 90% confidence that system #2 is better than system #1.

Extra Credit

If sampling from a data set with values that are self-similar or long range dependent (LRD), does the Central Limit Theorem apply? Explain why or why not. Explain how to determine the run time for a model that has a workload that is self-similar (for example, assume that job sizes are LRD for a disk simulation).

It does not apply. CLT only applies to samples from a population with values that have finite mean and variance (that is, the distribution for the population has finite mean and variance). If the workload to a simulation model is self-similar (that is with infinite, or extremely large, mean and variance) then the model will never converge to a steady-state (there is not steady state) and one can never say that a given run time is "long enough" (as a slightly longer run time could change the results significant if an extreme event occurs).

Please read and sign the following if true

All the assignments that I have submitted (and the class project to be submitted) for this class have been completed with software which I have legally acquired and for which I meet all licensing requirements. If requested I can and will produce a purchase receipt, or other evidence of legal acquisition and use.

Name (signature)

Date

Appendix A

A random variable uniformly distributed (continuous) in $a \leq x \leq b$ has probability density function,

$$f(x) = \begin{cases} \frac{1}{b-a} & a \leq x \leq b \\ 0 & \text{otherwise.} \end{cases}$$

A random variable Poisson distributed for a rate λ of arrivals has probability mass function,

$$f(k) = \frac{\lambda^k}{k!} e^{-\lambda} \quad k=0,1,2,\dots$$

A random variable exponentially distributed for rate λ has density function,

$$f(t) = \begin{cases} \lambda e^{-\lambda t} & t > 0 \\ 0 & t \leq 0 \end{cases}$$

and distribution function,

$$F(t) = \begin{cases} 1 - e^{-\lambda t} & x > 0 \\ 0 & x \leq 0. \end{cases}$$

Appendix B

```
// ex3_2_4.c
#include "csim.h"
#include <stdio.h>

void function1(void);

void sim(void)
{
    int i;
    create("sim");
    printf("Begin at time = %f \n", clock);
    function1();
    for (i=0; i<5; i++)
    {
        printf("sim() at %f \n", clock);
        hold(1.0);
    }
    printf("End at time = %f \n", clock);
}

void function1()
{
    int i;
    create("function1");
    for (i=0; i<3; i++)
    {
        hold(3.0);
        printf("function1() at %f \n", clock);
    }
}
```

Appendix C

6 points for changes in sim() and 6 points for changes in queue1(). Note that queue1() changes could be made in generate() as well. The "num =" line is the most important – it is worth 3 points alone.

```
#include "csim.h"
#include <stdio.h>

#define SIM_TIME 1.0e5

FACILITY Server;

int OddCount, EvenCount;

void generate(double lambda, double mu);
void queue1(double service_time);

void sim(void)
{
    double lambda, mu;

    create("sim");

    Server = facility("Server");
    lambda = 1.0;
    mu = 3.0;

    generate(lambda, mu);
    hold(SIM_TIME);

    report();

    printf("Arrivals that see an odd number of customers is %f %% \n",
        100.0 * OddCount / (OddCount + EvenCount));
    printf("Arrivals that see an even number of customers is %f %% \n",
        100.0 * EvenCount / (OddCount + EvenCount));
}

void generate(double lambda, double mu)
{
    double interarrival_time;
    double service_time;

    create("generate");

    while(1)
    {
        interarrival_time = exponential(1.0 / lambda);
        hold(interarrival_time);

        service_time = exponential(1.0 / mu);
        queue1(service_time);
    }
}

void queue1(double service_time)
{
    int num;

    create("queue1");

    num = qlength(Server) + num_busy(Server);
    if ((num % 2) == 0)
        EvenCount++;
    else
        OddCount++;

    reserve(Server);
    hold(service_time);
    release(Server);
}
```

Appendix D

```
#include "csim.h"
#include <stdio.h>

#define SIM_TIME 1.0e5
```

```
FACILITY Server1;
FACILITY Server2;
```

```
void generate(double lambda, double mu1, double mu2);
void queue1(double service_time, double mu2);
void queue2(double service_time);
```

```
void sim(void)
{
    double    lambda, mu1, mu2;

    create("sim");

    Server1 = facility("Server #1");
    Server2 = facility("Server #2");
    lambda = 1.0;
    mu1 = 3.0;
    mu2 = 2.0;

    generate(lambda, mu1, mu2);
    hold(SIM_TIME);

    report();
    printf("Delay in queue #1 = %f sec \n", resp(Server1));
    printf("Delay in queue #2 = %f sec \n", resp(Server2));
    printf("Utilization of server in queue #1 = %f %% \n",
        100.0 * util(Server1));
    printf("Utilization of server in queue #2 = %f %% \n",
        100.0 * util(Server2));
    printf("E2E delay = %f sec \n", resp(Server1) + resp(Server2));
}
```

```
void generate(double lambda, double mu1, double mu2)
{
    double    interarrival_time;
    double    service_time;

    create("generate");

    while(1)
    {
        interarrival_time = exponential(1.0 / lambda);
        hold(interarrival_time);
        service_time = exponential(1.0 / mu1);
        queue1(service_time, mu2);
    }
}
```

```
void queue1(double service_time, double mu2)
{
    create("queue1");

    reserve(Server1);
    hold(service_time);
    release(Server1);

    service_time = 1.0 / mu2;
    queue2(service_time);
}
```

```
void queue2(double service_time)
{
    create("queue2");
    reserve(Server2);
    hold(service_time);
    release(Server2);
}
```

6 points for changes in sim() and 6 points for changes in processes. It is possible to use a table to record and get the e2e delay.

Appendix E

4 points for the counter operation, 4 points for the if-check, and 4 points for describing the new variables.

```
//=====
//== Process to serve read requests ==
//=====
void disk(double file_size, double org_time)
{
    create("disk");

    // Increment the total number of read requests
    Total_req++;

    // Tally larger-than-cache files
    if (file_size > Cache_size)
        TooBigForCache++;

    // Process the file read request
    reserve(Disk_facility);
    if ((uniform(0.0, 1.0) < Prob_hit) && (file_size <= Cache_size))
    {
        Cache_hit++; // Increment the cache hit counter
        hold(file_size / Read_rate1); // Read the file bytes from cache
    }
    else
    {
        hold(uniform(0.0, Seek_time)); // Move the head
        hold(uniform(0.0, Spin_time)); // Spin the disk to position
        hold(file_size / Read_rate2); // Read the file bytes from disk
    }
    release(Disk_facility);

    // Record the response time
    record((clock - org_time), Resp_table);
}
```

Two new global variables are Cache_size initialized to the cache size in bytes and TooBigForCache counter initialized to zero.