

>>> Assignment #4 for Simulation (CIS 4930) <<<

>>> SOLUTIONS <<<

This assignment covers material from the fourth week of class lecture.

Problem #1 (35 points)

Determine X , Ts , U , W , Wq , L , and Lq for the following single-server queueing system for the time period 0 to 150 seconds. Carefully show your work including all pertinent figures and formulas. **Hint:** Review your week #4 reading (MacDougall, Chapter 1).

- Arrival #1 at time = 10 seconds with service time = 20 seconds
- Arrival #2 at time = 20 seconds with service time = 30 seconds
- Arrival #3 at time = 35 seconds with service time = 10 seconds
- Arrival #4 at time = 80 seconds with service time = 120 seconds
- Arrival #5 at time = 100 seconds with service time = 20 seconds

For solution see next page (scan of handwritten solution)

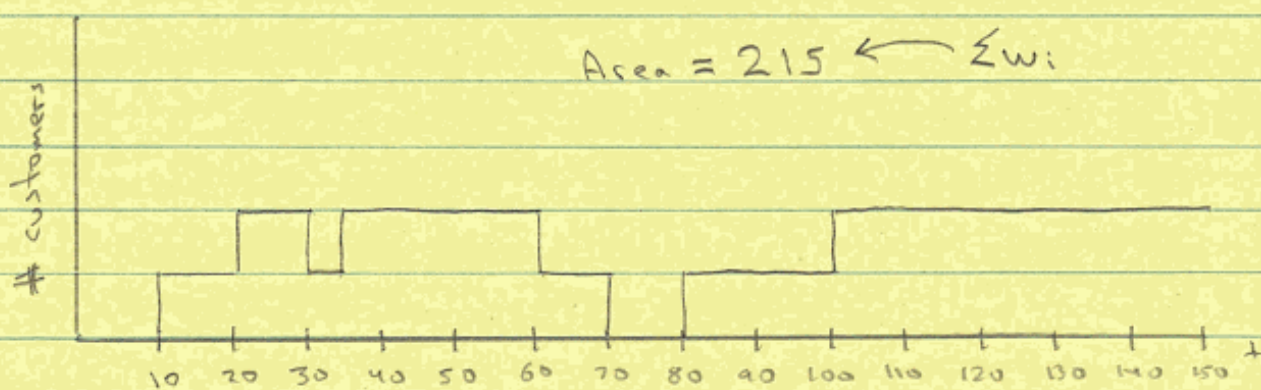
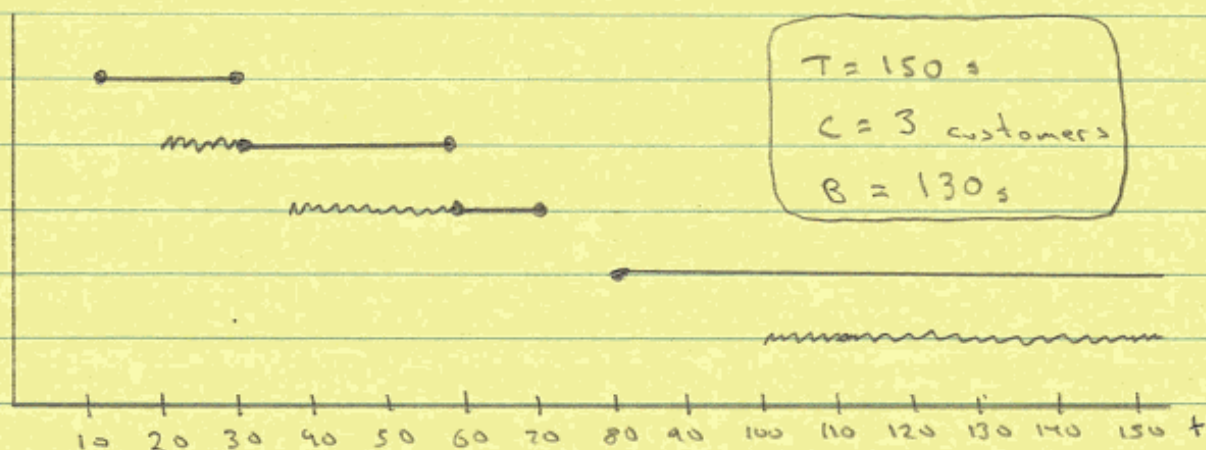
5 points for arrivals figure, 10 points for number in system figure, 15 points for right formulas (2 pts each), and 5 points for numerical results.

V1

KJC

6/17/11

Solution to Problem #2



$$\lambda = C/T = 3/150 \text{ customers/s}$$

$$T_s = B/C = 130/3 \text{ s}$$

$$U = B/T = 130/150$$

$$W = \sum w_i / C = 215/3 \text{ s}$$

$$W_q = W - T_s = 215/3 - 130/3 = 85/3 \text{ s}$$

$$L = \sum w_i / T = 215/150 \text{ customers}$$

$$L_q = L - U = 215/150 - 130/150 = 85/150$$

Problem #2 (30 points)

15 points for each graph and discussion – results must be identical to below.

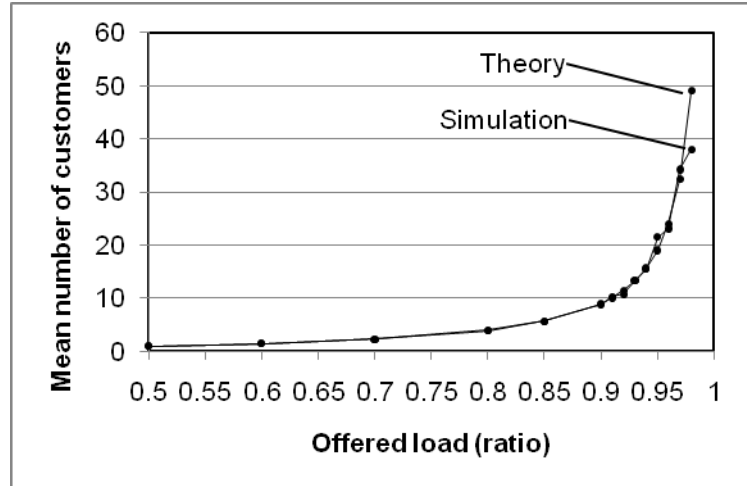
Using the mm1.c simulation program we discussed in class (and that is available for download via the class website), simulate the following offered loads for an M/M/1 queue: 50%, 60%, 70%, 80%, 85%, 90%, 91%, 92%, ..., 98%. For each offered load collect results on the mean number of customers in the system (L). Use a SIM_TIME of 200000 seconds. Plot both the simulation results and theory results (based on the formula for L for M/M/1) on one graph. Plot a graph of relative error for simulation to theory versus offered load on another graph. Comment on the relative error. Does it stay the same for all offered loads?

M/M/1 simulation and theory results

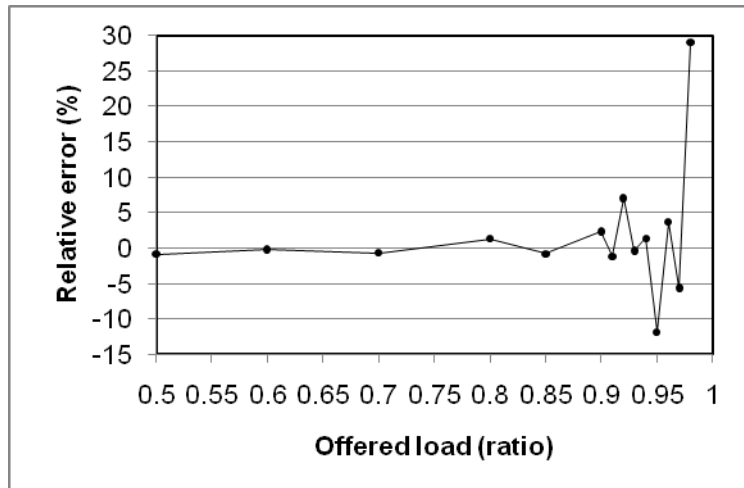
KJC (mm1.xls - 06/06/11)

SIM_TIME = 200000 seconds

offered	sim L	theory L	Error (%)
0.5	1.008	1.000	-0.82
0.6	1.503	1.500	-0.21
0.7	2.349	2.333	-0.67
0.8	3.948	4.000	1.31
0.85	5.713	5.667	-0.82
0.9	8.793	9.000	2.36
0.91	10.231	10.111	-1.17
0.92	10.744	11.500	7.04
0.93	13.337	13.286	-0.38
0.94	15.453	15.667	1.38
0.95	21.541	19.000	-11.80
0.96	23.153	24.000	3.66
0.97	34.251	32.333	-5.60
0.98	37.980	49.000	29.02



The absolute magnitude of relative error increases as offered load



Problem #3 (35 points)

5 points for model change, 15 points for each graph and discussion – results must be identical to below.

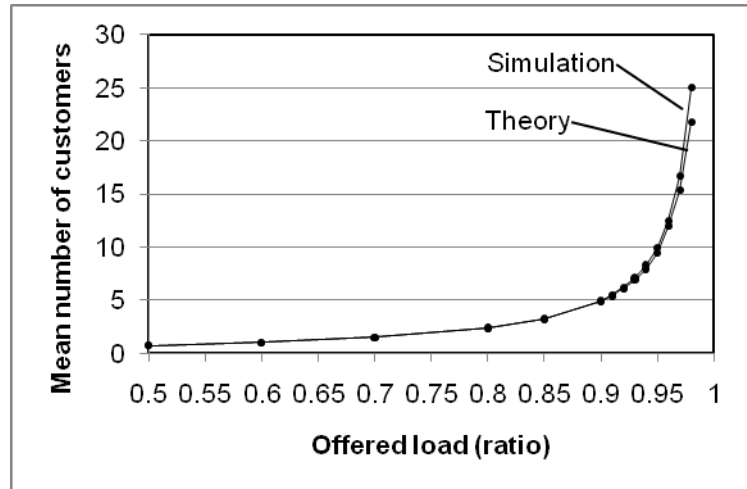
Repeat problem #1 for M/D/1 (of course, you can't use the formula for L for M/M/1, you must use the P-K formula correctly). You will need to modify `mm1.c` to model an M/D/1 queue. In addition to the two plots, also submit your modified `mm1.c` (perhaps call it `md1.c`?) source code. Comments on the relative error – is it greater or smaller than for the M/M/1 simulation? Speculate on the “why”.

M/D/1 simulation and theory results

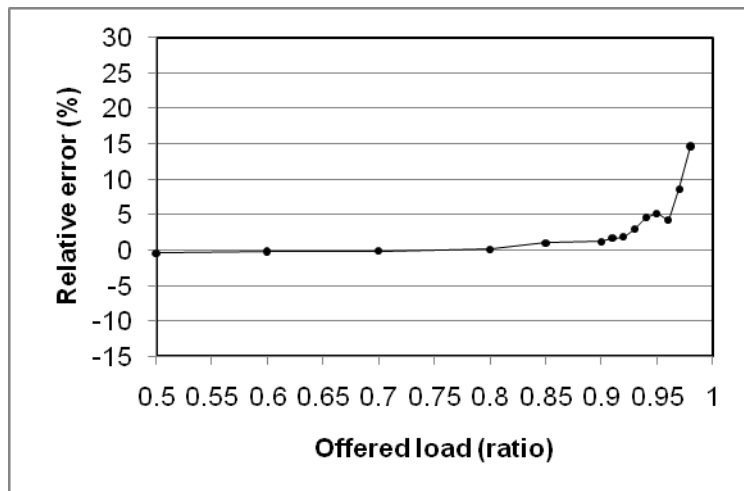
KJC (md1.xls - 06/06/11)

SIM_TIME = 200000 seconds

offered	sim L	theory L	Error (%)
0.5	0.753	0.750	-0.45
0.6	1.052	1.050	-0.18
0.7	1.518	1.517	-0.12
0.8	2.398	2.400	0.10
0.85	3.224	3.258	1.05
0.9	4.891	4.950	1.20
0.91	5.421	5.511	1.65
0.92	6.096	6.210	1.88
0.93	6.906	7.108	2.93
0.94	7.938	8.303	4.60
0.95	9.482	9.975	5.20
0.96	11.977	12.480	4.20
0.97	15.327	16.652	8.65
0.98	21.793	24.990	14.67



The absolute magnitude of relative error is less than for the M/M/1 simulation. This is because there is less variability in the M/D/1 case (deterministic service, which has variance = 0) than the M/M/1 case (exponential service time with non-zero variance).



Appendix – M/D/1 simulation program for problem #3

```
//===== file = mdl.c =====
//= A simple "straight C" M/D/1 queue simulation =
//=====
//= Notes: =
//= 1) The values of SIM_TIME, ARR_TIME, and SERV_TIME need to be set. =
//-----
//= Build: gcc mdl.c -lm, bcc32 mdl.c, cl mdl.c =
//-----
//= Execute: mml =
//-----
//= History: KJC (06/06/11) - Genesis (from mml.c) =
//=====
//----- 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 200000.0 // Simulation time
#define ARR_TIME (1.0/0.90) // 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
    double 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 (time < end_time)
    {
        if (t1 < t2) // *** Event #1 (arrival) ***
        {
```

```

time = t1;
s = s + n * (time - tn); // 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 + Ts; // NOTE: This line is modified from mml.c
}
}
else // *** Event #2 (departure) ***
{
    time = t2;
    s = s + n * (time - tn); // Update area under "s" curve
    n--;
    tn = time; // tn = "last event time" for next event
    c++; // Increment number of completions
    if (n > 0)
        t2 = time + Ts; // NOTE: This line is modified from mml.c
    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/D/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);
}

```

```

//=====
//= 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));
}

```