CSC 641

Server Simulations G/G/1

Richard Robinson

October 27, 2017

Professor: Jozo Dujmovic

Statement of Problem:

The goal of this program is to create a simulation of 9 different possible combinations for a single server system. These combinations are made through arrival times and service times that are either constant, uniform, or exponential.

The averages for response time, service time, interarrival time, waiting time, and utilization will be found.

After the simulation is completed the results should be compared to calculated values gained using G/G/1 equations.

Program Details:
I used Kingman's formula for wait times, and generated response time from expanding upon that formula in the calculations. For calculating variation I used 1/12 * (upper bound – lower bound)^2 for uniform distribution and 1 for exponential distribution (this confuses me because I thought the variation for exponential would be λ^{-2} = mean^2. The variation of a constant distribution is of course 0. I did not use the equations given to us in class, as these formulae used were more accurate for the specific components within this program.

For the simulation I calculated the arrival time for the next arrival, the next service time, and serviced an item if present or waited for an item to arrive if no job was there. This program works by waiting for the next event to occur-- if an arrival occurs before servicing time passes, current time remaining on service is decremented, the incoming arrival has its time pushed upon the stack, and the next arrival time is generated. If a service finishes before the next arrival if there is another item in queue it is popped, time is incremented, the next service time is generated, and the time until next arrival is decremented by the amount of service time that passed.

I only used one queue which contained the times that the arrival would be serviced at if there was no queue present. For generating response times, I subtracted this time from the time at which that element finished being serviced. For average queue length whenever time passed I multiplied the wait

The program terminates when a pre-defined amount of services occur.

Program Structure:

Driver.cpp:
The Driver for the program, creates a simulation for each one of the 9 cases and runs them.

RandomNumbers.cpp:
Generates methods for uniform, exponential, and constant random number generation.
I gave all methods the same parameters and made one for constants so that I could make a pointer to functions in my SimulateServer class with the same parameters instead of using many different cases.

ServerSimulation.cpp:
Contains the SimulateServer class, which runs a simulation of a server.

I did not make a header file this time because of time constraints, so the code is not as organized.

This class contains methods to store and find the mean and variance of service time, inter-arrival time, response time, queue length, and server utilization.

These are computed using event driven methods.

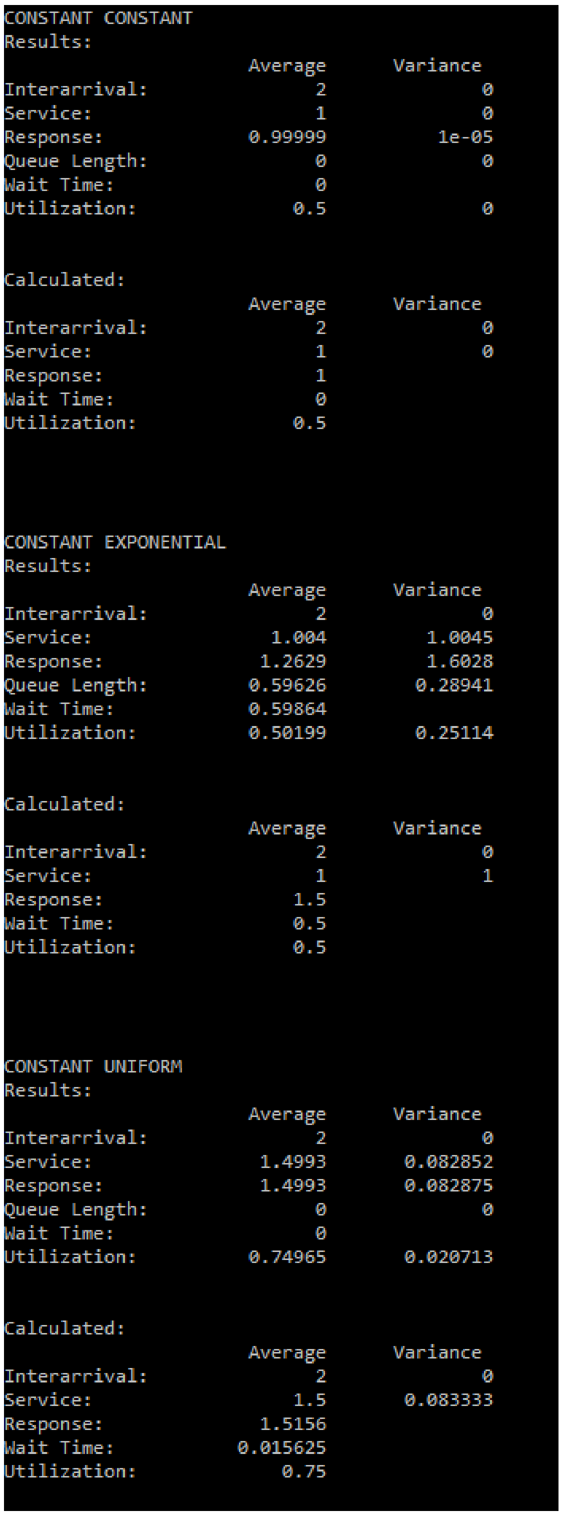
MathCalculationsForServer.cpp
This includes the calculations of values to have the simulation compared to.

Results:
The results are shown on the following pages, the variance for exponential calculations was 1 for response time calculation formula given and $\beta^2 = \lambda^{-2} = \text{mean}^2$ for the actual variance shown in the charts.

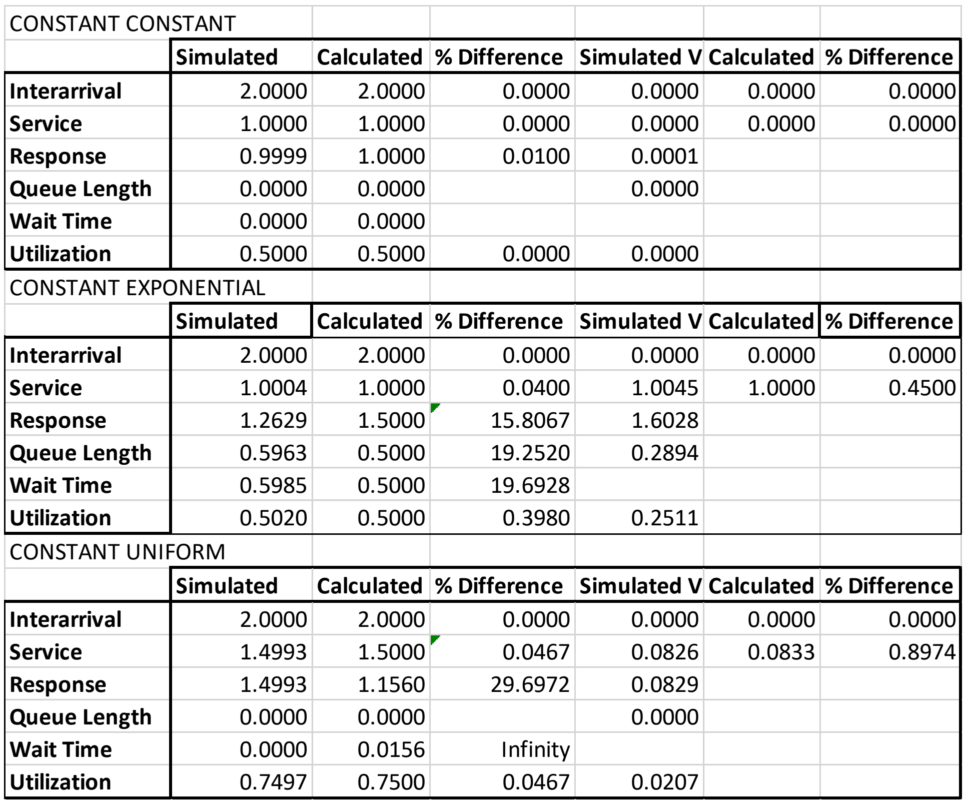
Simulated V is the variance of the simulation and the Calculated after that is the calculated variance.

All times are in seconds.

The notation for the type of interarrival type / service type pair is given by
<INTERARRIVAL TYPE> < SERVICE TYPE>



```
CONSTANT CONSTANT
Results:
                              Average         Variance
Interarrival:                       2                0
Service:                            1                0
Response:                     0.99999            1e-05
Queue Length:                       0                0
Wait Time:                          0
Utilization:                      0.5                0


Calculated:
                              Average         Variance
Interarrival:                       2                0
Service:                            1                0
Response:                           1
Wait Time:                          0
Utilization:                      0.5




CONSTANT EXPONENTIAL
Results:
                              Average         Variance
Interarrival:                       2                0
Service:                        1.004           1.0045
Response:                      1.2629           1.6028
Queue Length:                 0.59626          0.28941
Wait Time:                    0.59864
Utilization:                  0.50199          0.25114


Calculated:
                              Average         Variance
Interarrival:                       2                0
Service:                            1                1
Response:                         1.5
Wait Time:                        0.5
Utilization:                      0.5




CONSTANT UNIFORM
Results:
                              Average         Variance
Interarrival:                       2                0
Service:                       1.4993         0.082852
Response:                      1.4993         0.082875
Queue Length:                       0                0
Wait Time:                          0
Utilization:                  0.74965         0.020713


Calculated:
                              Average         Variance
Interarrival:                       2                0
Service:                          1.5         0.083333
Response:                      1.5156
Wait Time:                   0.015625
Utilization:                     0.75
```



CONSTANT CONSTANT

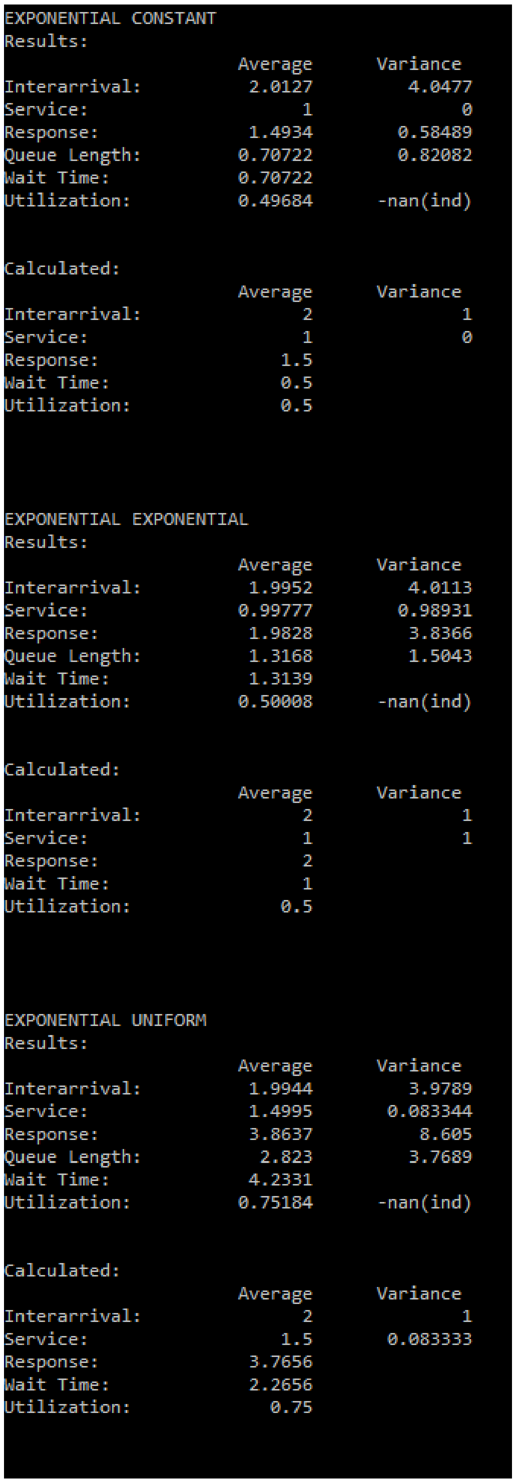
	Simulated	Calculated	% Difference	Simulated V	Calculated	% Difference
Interarrival	2.0000	2.0000	0.0000	0.0000	0.0000	0.0000
Service	1.0000	1.0000	0.0000	0.0000	0.0000	0.0000
Response	0.9999	1.0000	0.0100	0.0001		
Queue Length	0.0000	0.0000		0.0000		
Wait Time	0.0000	0.0000				
Utilization	0.5000	0.5000	0.0000	0.0000		

CONSTANT EXPONENTIAL

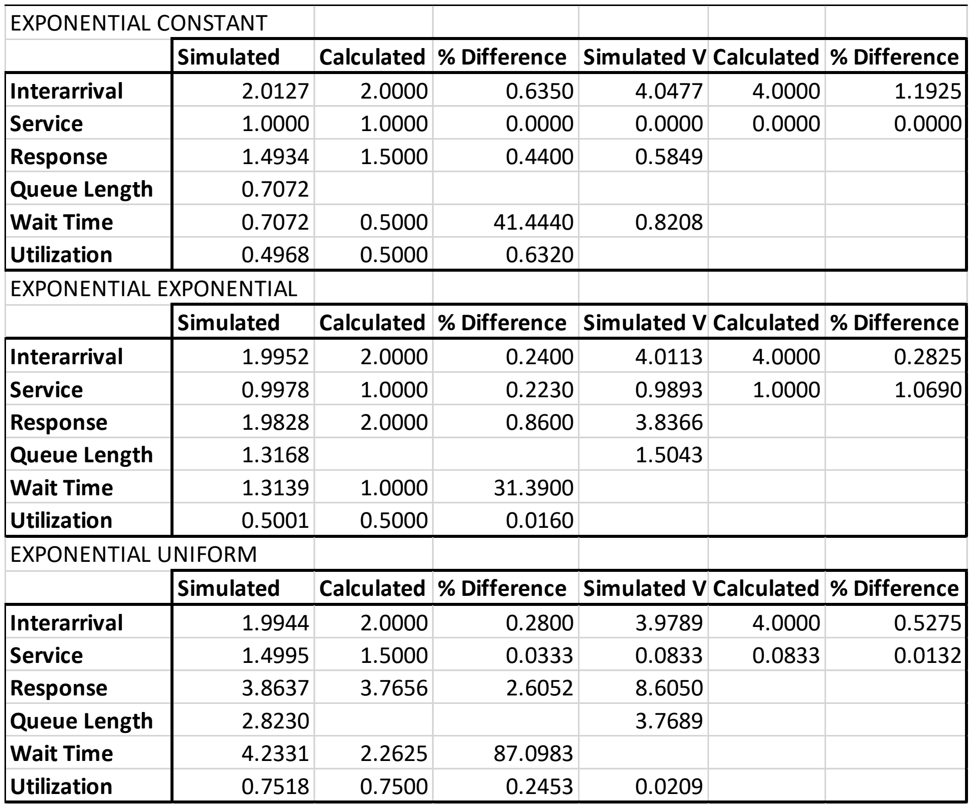
	Simulated	Calculated	% Difference	Simulated V	Calculated	% Difference
Interarrival	2.0000	2.0000	0.0000	0.0000	0.0000	0.0000
Service	1.0004	1.0000	0.0400	1.0045	1.0000	0.4500
Response	1.2629	1.5000	15.8067	1.6028		
Queue Length	0.5963	0.5000	19.2520	0.2894		
Wait Time	0.5985	0.5000	19.6928			
Utilization	0.5020	0.5000	0.3980	0.2511		

CONSTANT UNIFORM

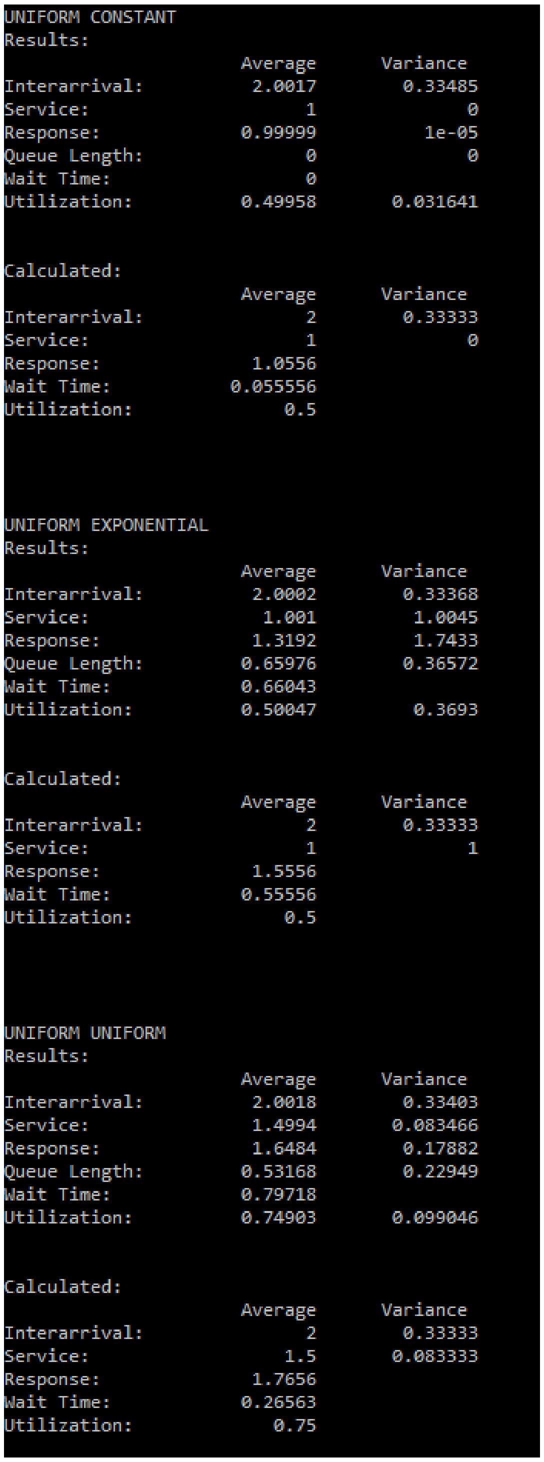
	Simulated	Calculated	% Difference	Simulated V	Calculated	% Difference
Interarrival	2.0000	2.0000	0.0000	0.0000	0.0000	0.0000
Service	1.4993	1.5000	0.0467	0.0826	0.0833	0.8974
Response	1.4993	1.1560	29.6972	0.0829		
Queue Length	0.0000	0.0000		0.0000		
Wait Time	0.0000	0.0156	Infinity			
Utilization	0.7497	0.7500	0.0467	0.0207		



```
EXPONENTIAL CONSTANT
Results:
                      Average       Variance
Interarrival:          2.0127         4.0477
Service:                    1              0
Response:              1.4934        0.58489
Queue Length:         0.70722        0.82082
Wait Time:            0.70722
Utilization:          0.49684       -nan(ind)


Calculated:
                      Average       Variance
Interarrival:               2              1
Service:                    1              0
Response:                 1.5
Wait Time:                0.5
Utilization:              0.5




EXPONENTIAL EXPONENTIAL
Results:
                      Average       Variance
Interarrival:          1.9952         4.0113
Service:              0.99777        0.98931
Response:              1.9828         3.8366
Queue Length:          1.3168         1.5043
Wait Time:             1.3139
Utilization:          0.50008       -nan(ind)


Calculated:
                      Average       Variance
Interarrival:               2              1
Service:                    1              1
Response:                   2
Wait Time:                  1
Utilization:              0.5




EXPONENTIAL UNIFORM
Results:
                      Average       Variance
Interarrival:          1.9944         3.9789
Service:               1.4995       0.083344
Response:              3.8637          8.605
Queue Length:           2.823         3.7689
Wait Time:             4.2331
Utilization:          0.75184       -nan(ind)


Calculated:
                      Average       Variance
Interarrival:               2              1
Service:                  1.5       0.083333
Response:              3.7656
Wait Time:             2.2656
Utilization:             0.75
```



EXPONENTIAL CONSTANT						
	Simulated	**Calculated**	**% Difference**	**Simulated V**	**Calculated**	**% Difference**
Interarrival	2.0127	2.0000	0.6350	4.0477	4.0000	1.1925
Service	1.0000	1.0000	0.0000	0.0000	0.0000	0.0000
Response	1.4934	1.5000	0.4400	0.5849		
Queue Length	0.7072					
Wait Time	0.7072	0.5000	41.4440	0.8208		
Utilization	0.4968	0.5000	0.6320			
EXPONENTIAL EXPONENTIAL						
	Simulated	**Calculated**	**% Difference**	**Simulated V**	**Calculated**	**% Difference**
Interarrival	1.9952	2.0000	0.2400	4.0113	4.0000	0.2825
Service	0.9978	1.0000	0.2230	0.9893	1.0000	1.0690
Response	1.9828	2.0000	0.8600	3.8366		
Queue Length	1.3168			1.5043		
Wait Time	1.3139	1.0000	31.3900			
Utilization	0.5001	0.5000	0.0160			
EXPONENTIAL UNIFORM						
	Simulated	**Calculated**	**% Difference**	**Simulated V**	**Calculated**	**% Difference**
Interarrival	1.9944	2.0000	0.2800	3.9789	4.0000	0.5275
Service	1.4995	1.5000	0.0333	0.0833	0.0833	0.0132
Response	3.8637	3.7656	2.6052	8.6050		
Queue Length	2.8230			3.7689		
Wait Time	4.2331	2.2625	87.0983			
Utilization	0.7518	0.7500	0.2453	0.0209		



```
UNIFORM CONSTANT
Results:
                      Average       Variance
Interarrival:          2.0017        0.33485
Service:                    1              0
Response:             0.99999          1e-05
Queue Length:               0              0
Wait Time:                  0
Utilization:          0.49958       0.031641


Calculated:
                      Average       Variance
Interarrival:               2        0.33333
Service:                    1              0
Response:              1.0556
Wait Time:           0.055556
Utilization:              0.5



UNIFORM EXPONENTIAL
Results:
                      Average       Variance
Interarrival:          2.0002        0.33368
Service:                1.001         1.0045
Response:              1.3192         1.7433
Queue Length:         0.65976        0.36572
Wait Time:            0.66043
Utilization:          0.50047         0.3693


Calculated:
                      Average       Variance
Interarrival:               2        0.33333
Service:                    1              1
Response:              1.5556
Wait Time:            0.55556
Utilization:              0.5



UNIFORM UNIFORM
Results:
                      Average       Variance
Interarrival:          2.0018        0.33403
Service:               1.4994       0.083466
Response:              1.6484        0.17882
Queue Length:         0.53168        0.22949
Wait Time:            0.79718
Utilization:          0.74903       0.099046


Calculated:
                      Average       Variance
Interarrival:               2        0.33333
Service:                  1.5       0.083333
Response:              1.7656
Wait Time:            0.26563
Utilization:             0.75
```

UNIFORM CONSTANT						
	Simulated	**Calculated**	**% Difference**	**Simulated V**	**Calculated**	**% Difference**
Interarrival	2.0017	2.0000	0.0850	0.3349	0.3333	0.4560
Service	1.0000	1.0000	0.0000	0.0000	0.0000	0.0000
Response	1.0000	1.0556	5.2672	0.0000		
Queue Length	0.0000			0.0000		
Wait Time	0.0000	0.0556	100.0000			
Utilization	0.4996	0.5000	0.0840	0.0316		
UNIFORM EXPONENTIAL						
	Simulated	**Calculated**	**% Difference**	**Simulated V**	**Calculated**	**% Difference**
Interarrival	2.0002	2.0000	0.0100	0.3337	0.3333	0.1050
Service	1.0010	1.0000	0.1000	1.0045	1.0000	0.0000
Response	1.3192	1.5556	15.1967	1.7433		
Queue Length	0.6598			0.3657		
Wait Time	0.6604	0.5556	18.8773			
Utilization	0.5005	0.5000	0.0940	0.3693		
UNIFORM UNIFORM						
	Simulated	**Calculated**	**% Difference**	**Simulated V**	**Calculated**	**% Difference**
Interarrival	2.0018	2.0000	0.0900	0.3340	0.3333	0.2100
Service	1.4994	1.5000	0.0400	0.0835	0.0833	0.0000
Response	1.6484	1.7656	6.6380	0.1788		
Queue Length	0.5317			0.2295		
Wait Time	0.7972	0.2656	200.1092			
Utilization	0.7490	0.7500	0.1293	0.0990		

Conclusions:

For interarrival times, service times, and response times the simulation matched well with the calculated values. The calculated wait times for exponential functions varied immensely from the simulated values, this is either because the equation used does not adequately model wait times for exponential functions or because my simulation method is flawed.

Most of results behaved as anticipated and the simulation matched well with the calculated values.

I would have preferred all the results to match up well, but it was not unexpected that there was one data field that was not adequately represented in both the simulation and the calculations as there was a large variety of arrival and service time distributions.

Additional Discussion:

The way I implemented the random number generator was poor and I have learned my lesson. I wanted to pass pointers to functions and just call one function depending on what type of data was present and made a function pointer to do so. This required the parameters to match so the value passed into it for uniform distribution was a range instead of the mean, where 1.2 would mean a range from 1 to 2 and 1.3 would mean a range from 1 to 3. This was an awkward implementation of this. Next time I will include more parameters or just use a hard-coded switch statement.

The code was not as well organized and planned out as I initially hoped. It is rather easily readable (to me at least), but I would have liked to break it down into smaller components. The methods are rather short, but there are many of them. Calculations were rather strange, I used Kingsman's formula instead of ones discussed in class and the variation for exponential numbers had to be done one way for the response method to work (the way described in this course where variance = 1) and another for the actual variance because it is most definitely not 1 for the exponential function I implemented.

Overall, this project was a somewhat positive experience. I made a few unsavory mistakes while progressing through the simulation, such as forgetting a += and instead doing = on a summation component, and dividing by the wrong counter.

There are likely some mistakes that I did make in places or another (explaining the large wait time %difference).

The results appear to be mostly correct, and I will definitely be able to apply knowledge and experience gained from this project to projects in the future.

Source:

ServerSimulation.cpp

```
#include <iostream>
#include <queue> //queue
#include <iomanip> //setw
#include "RandomNumbers.cpp" //number generation

using namespace std;

//Number of items to process
static const int TOTAL_NUMBER_SERVICES = 100000;

class SimulateServer
{
private:
	char *arrivalType, *serviceType;
	double arrivalMean, serviceMean;
	double totalTime = 0;

	//Storing results to find std dev
	double serviceTimeArr[TOTAL_NUMBER_SERVICES];
	double interArrivalTimeArr[TOTAL_NUMBER_SERVICES * 2];
	double responseTimeArr[TOTAL_NUMBER_SERVICES * 2];
	double queueArr[TOTAL_NUMBER_SERVICES];
	double serverUtilizationArr[TOTAL_NUMBER_SERVICES];

	//Std dev
	double varService, varInterarrival, varResponse, varQueue, varUtilization,
varWaitTime;



	//values needed for results

	//TOTALS
	double totalInterarrivalTime = 0, totalServiceTime = 0, totalResponseTime = 0,
serverUtilizationFactor = 0, queueFactor = 0;

	//AVERAGES
	double avgInterarrivalTime, avgServiceTime, avgResponseTime, avgUtilization,
avgQueue, avgWaitTime;

	//RESULTS
	double timeToNextArrival, currentServiceRemaining;
	double nextArrivalTime;

	//Pointers to functions
	double (*arrivalGeneration)(double value);
	double(*serviceGeneration)(double value);

	//Counter for trials
	int serviceCounter = 0;
	int arrivalCounter = 0;
```

```
        //Current Queue
        queue <double> simulatorQueue;


        void resetData()
        {
                totalTime = 0;
                totalInterarrivalTime = 0;
                totalServiceTime = 0;
                totalResponseTime = 0;
                serverUtilizationFactor = 0;
                queueFactor = 0;
        }

        void waitForArrival()
        {
               totalTime += timeToNextArrival;                        //TIME
               arrive();
        }


        void getNextArrival()  //GENERATE NEXT ARRIVAL DATA
        {
               timeToNextArrival = arrivalGeneration(arrivalMean);
               nextArrivalTime = totalTime + timeToNextArrival;
               totalInterarrivalTime += timeToNextArrival;                   //TOTAL
Interarrival time for all arrival times generated
               arrivalCounter++;
               interArrivalTimeArr[arrivalCounter - 1] = timeToNextArrival;
        }

        void getNextService() //GENERATE NEXT SERVICE
        {
               serviceCounter++;
               currentServiceRemaining = serviceGeneration(serviceMean); //GENERATE NEXT
SERVICE TIME
               totalServiceTime += currentServiceRemaining;

               serviceTimeArr[serviceCounter -1] = currentServiceRemaining;
        }

        void arrive() //PUSH, GENERATE NEXT ARRIVAL
        {
               queueArr[serviceCounter - 1] = simulatorQueue.size();
               simulatorQueue.push(nextArrivalTime );
               getNextArrival();
        }

        void arriveWhileServicing()
        {
               currentServiceRemaining = currentServiceRemaining - timeToNextArrival;
//decrement service time
               totalTime += timeToNextArrival;          //THIS MUCH TIME PASSED
               queueFactor += (double)( simulatorQueue.size() ) * timeToNextArrival;
//Every time passes need to do this
               arrive();
        }
```

```
	void completeService()
	{
		//CHECK THIS LATER
		totalTime += currentServiceRemaining; //THIS MUCH TIME PASSED
		totalResponseTime += totalTime - simulatorQueue.front(); //Response time is 
found as dif between arrival and working on
		responseTimeArr[serviceCounter - 1] = totalTime - simulatorQueue.front();
		simulatorQueue.pop();

		timeToNextArrival = timeToNextArrival - currentServiceRemaining;  //find 
time to next arrival

		queueFactor += (double)( simulatorQueue.size() ) * currentServiceRemaining; 
//every time passes need to do this
	}

	void service()
	{
		{
			//IF NEXT JOB ARRIVES BEFORE CURRENT SERVICE FINISHES
			if (timeToNextArrival < currentServiceRemaining)  { 
arriveWhileServicing(); }
			//WORK ON NEXT ITEM
			else
			{
				completeService();
				getNextService();
			}
		}
	}

	void generateAverages()
	{
		avgInterarrivalTime = totalInterarrivalTime / arrivalCounter;
		avgServiceTime = totalServiceTime / serviceCounter;
		avgResponseTime = totalResponseTime / serviceCounter;
		avgUtilization =  avgServiceTime / avgInterarrivalTime;
		avgQueue = queueFactor / serviceCounter;
		avgWaitTime = avgQueue * avgServiceTime;
	}

	double findVariance( double *item)
	{
		double tempArr[TOTAL_NUMBER_SERVICES];
		double tempSum = 0;
		double tempMean;
		double sumSquareDifference = 0;
		for (int i = 0; i < TOTAL_NUMBER_SERVICES; i++)
		{
			tempArr[i] = item[i];
			tempSum += tempArr[i];
		}
		tempMean = tempSum / TOTAL_NUMBER_SERVICES;

		for (int i = 0; i < TOTAL_NUMBER_SERVICES; i++)
		{
			sumSquareDifference += (tempArr[i] - tempMean) * (tempArr[i] - 
tempMean);
```

```
		}
		return sumSquareDifference / (TOTAL_NUMBER_SERVICES - 1);
		//return pow( (sumSquareDifference / (TOTAL_NUMBER_SERVICES - 1)), .5);
	}

	void calculateVariance()
	{
		for (int i = 0; i < TOTAL_NUMBER_SERVICES; i++)
		{
			serverUtilizationArr[i] = serviceTimeArr[i] /
interArrivalTimeArr[i];
		}
		varUtilization = findVariance(serverUtilizationArr);
		varInterarrival = findVariance(interArrivalTimeArr);
		varService = findVariance(serviceTimeArr);
		varResponse = findVariance(responseTimeArr);
		varQueue = findVariance(queueArr);
	}

public:
	SimulateServer(char *arrivalType, double arrivalMean, char *serviceType, double
serviceMean)
	{
		this->arrivalType = arrivalType;
		this->arrivalMean = arrivalMean;
		this->serviceType = serviceType;
		this->serviceMean = serviceMean;
	}

	void simulate()
	{
		//Generate functions to be used for indicated server
		if (arrivalType == "uniform") arrivalGeneration = generateUniform;
		else if (arrivalType == "exponential") arrivalGeneration =
generateExponential;
		else if (arrivalType == "constant") arrivalGeneration = generateConstant;

		if (serviceType == "uniform") serviceGeneration = generateUniform;
		else if (serviceType == "exponential") serviceGeneration =
generateExponential;
		else if (serviceType == "constant") serviceGeneration = generateConstant;


		//Generate initial values
		resetData();
		getNextArrival();
		getNextService();


		while (serviceCounter < TOTAL_NUMBER_SERVICES)
		{
			if (simulatorQueue.empty())  waitForArrival();
			service();
		}

		generateAverages();
		calculateVariance();
		printResults();
```

```
        }

        void printResults()
        {
                cout << "Results:\n";
                cout << setw(29) << setprecision(5) << setfill(' ') << "Average";
                cout << setw(15) << setprecision(5) << setfill(' ') << "Variance\n";
                cout << "Interarrival: ";
                cout << setw(15) << setprecision(5) << setfill(' ') << avgInterarrivalTime;
                cout << setw(15) << setprecision(5) << setfill(' ') << varInterarrival << 
"\n";
                cout << "Service:      ";
                cout << setw(15) << setprecision(5) << setfill(' ') << avgServiceTime;
                cout << setw(15) << setprecision(5) << setfill(' ') << varService << "\n";
                cout << "Response:     ";
                cout << setw(15) << setprecision(5) << setfill(' ') << avgResponseTime;
                cout << setw(15) << setprecision(5) << setfill(' ') << varResponse << "\n";
                cout << "Queue Length: ";
                cout << setw(15) << setprecision(5) << setfill(' ') << avgQueue;
                cout << setw(15) << setprecision(5) << setfill(' ') << varQueue << "\n";
                cout << "Wait Time:    ";
                cout << setw(15) << setprecision(5) << setfill(' ') << avgWaitTime;
                cout << setw(15) << setprecision(5) << setfill(' ') << "" << "\n";
                cout << "Utilization:  ";
                cout << setw(15) << setprecision(5) << setfill(' ') << avgUtilization;
                cout << setw(15) << setprecision(5) << setfill(' ') << varUtilization << 
"\n\n\n";

        }
};
```

RandomNumbers.cpp

```
#include <math.h>
#include <cstdlib>
#include <time.h>
static int numbersGenerated = 0;

static double generateUniform( double combination )
{

	double lowerBound = floor(combination);
	double upperBound = (double) (combination - (double)lowerBound)  * (double)10;
	double difference =  (upperBound - lowerBound);

	return (double)rand() / (double)RAND_MAX * (double)difference + lowerBound;
}

static double generateExponential( double mean )
{
	return -1.0 * mean * log(1.0 + 1.0/(double)RAND_MAX -
(double)rand()/(double)RAND_MAX);
}

static double generateConstant(double mean)
{
	return (double)mean;
}
```

MathCalculationsForServers.cpp

```
#include <iostream>
#include <iomanip>
using namespace std;


static char *serviceType, *arrivalType;
static double serviceMean, arrivalMean;
static double utilization, serviceVariation, arrivalVariation, responseTime, jobs, wait;



/*=============================================
*findVariation
*finds variation for types present
===========================================*/
static double findVariation( char *dataType, double mean, double lowerBound, double
upperBound )
{
	if (dataType == "constant") return 0;
	else if (dataType == "uniform")
	{
		//return (upperBound - lowerBound) / ((pow(3, .5) * (upperBound +
lowerBound)));
		return 1.0 / 12.0 * pow(upperBound - lowerBound, 2);
	}

	else if (dataType == "exponential")
	{
		return 1;
		//return mean*mean;
	}

	return 0;
}



/*=============================================
* findUtilization
* finds the utilization
===========================================*/
static double findUtilization( double avgServiceTime, double avgInterarrivalTime)
{
	utilization = avgServiceTime/ avgInterarrivalTime;
	return utilization;
}

/*=============================================
* findResponseTime
* finds the response time using G/G/1
===========================================*/
static double findResponseTime( double avgService, double variationService, double
utilization, double variationArrival)
```

```
{
	/*
	static double outside = avgService / (1.0 - utilization);
	static double numerator = (variationService * variationService + 1.0) * 
(variationArrival * variationArrival - 1.0);
	static double denominator = (utilization * utilization * variationService * 
variationService + 1.0);
	static double inside = 1.0 - utilization / 2.0 * (1.0 - variationService * 
variationService - numerator / denominator);
	responseTime = inside*outside;
	return responseTime;
	*/

}

/*===========================================
*findJobs
===========================================*/
static double findJobs( double utilization )
{
	jobs = utilization / (1.0 - utilization);
	return jobs;
}

/*===========================================
*findWait
===========================================*/
static double findWait()
{
    wait = utilization / (1 - utilization) * (pow(serviceVariation, 2) + 
pow(arrivalVariation, 2)) / 2 * serviceMean;
	return wait;
}

/*==============================================
*findCalculations
*finds all items
=============================================*/
static void findCalculations(char *inArrivalType, double inArrivalMean, char 
*inServiceType, double inServiceMean, double arrivalLowerBound, double arrivalUpperBound, 
double serviceLowerBound, double serviceUpperBound )
{
	serviceType = inServiceType;
	serviceMean = inServiceMean;
	arrivalType = inArrivalType;
	arrivalMean = inArrivalMean;


	arrivalVariation = findVariation( arrivalType, arrivalMean, arrivalLowerBound, 
arrivalUpperBound);
	serviceVariation = findVariation(serviceType, serviceMean, serviceLowerBound, 
serviceUpperBound);
	findUtilization(serviceMean, arrivalMean);
	responseTime = serviceMean + findWait();
	//findResponseTime(serviceMean, serviceVariation, utilization, arrivalVariation);
	findJobs(utilization);
}
```

```
/*============================================
*printResults
*prints results
==========================================*/
static void printResults()
{
       cout << "Calculated:\n";
       cout << setw(29) << setprecision(5) << setfill(' ') << "Average";
       cout << setw(15) << setprecision(5) << setfill(' ') << "Variance\n";
       cout << "Interarrival: ";
       cout << setw(15) << setprecision(5) << setfill(' ') << arrivalMean;
       cout << setw(15) << setprecision(5) << setfill(' ') << arrivalVariation << "\n";
       cout << "Service:      ";
       cout << setw(15) << setprecision(5) << setfill(' ') << serviceMean;
       cout << setw(15) << setprecision(5) << setfill(' ') << serviceVariation << "\n";
       cout << "Response:     ";
       cout << setw(15) << setprecision(5) << setfill(' ') << responseTime;
       cout << setw(15) << setprecision(5) << setfill(' ') << "" << "\n";
       cout << "Wait Time:    ";
       cout << setw(15) << setprecision(5) << setfill(' ') << wait;
       cout << setw(15) << setprecision(5) << setfill(' ') << "\n";
       cout << "Utilization:  ";
       cout << setw(15) << setprecision(5) << setfill(' ') << utilization;
       cout << setw(15) << setprecision(5) << setfill(' ') << "" << "\n\n\n";
}
```

Driver.cpp

```
#include <iostream>
#include "ServerSimulation.cpp"
#include "MathCalculationsForServers.cpp"
#include <queue>
using namespace std;

int main()
{

        cout << "Goodbye World\n";
        queue <double> testQ;

        cout << "CONSTANT CONSTANT\n";

        SimulateServer *simulateServer;
        //CONSTANT ARRIVAL ( 2 sec )
        //CONSTANT SERVICE ( 1 sec )

        simulateServer = new SimulateServer("constant", 2, "constant", 1);
        simulateServer->simulate();

        findCalculations("constant", 2, "constant", 1, 0, 0, 0, 0);
        printResults();

        cout << "\n\nCONSTANT EXPONENTIAL\n";
        //CONSTANT ARRIVAL ( 2 sec )
        //EXPONENTIAL SERVICE ( mean = 1 sec )
        simulateServer = new SimulateServer("constant", 2, "exponential", 1);
        simulateServer->simulate();

        findCalculations("constant", 2, "exponential", 1, 0, 0, 0, 0 );
        printResults();

        cout << "\n\nCONSTANT UNIFORM\n";
        //CONSTANT ARRIVAL ( 2 sec )
        //UNIFORM SERVICE ( 1 - 2 sec )
        simulateServer = new SimulateServer("constant", 2, "uniform", 1.2);
        simulateServer->simulate();

        findCalculations("constant", 2, "uniform", 1.5, 0, 0, 1, 2);
        printResults();

        cout << "\n\nEXPONENTIAL CONSTANT\n";
        //EXPONENTIAL ARRIVAL ( mean = 2 sec )
        //CONSTANT SERVICE ( 1 sec )
        simulateServer = new SimulateServer("exponential", 2, "constant", 1);
        simulateServer->simulate();

        findCalculations("exponential", 2, "constant", 1, 0, 0, 0, 0);
        printResults();

        cout << "\n\nEXPONENTIAL EXPONENTIAL\n";
        //EXPONENTIAL ARRIVAL ( mean = 2 sec )
        //EXPONENTIAL SERVICE ( mean = 1 sec )
        simulateServer = new SimulateServer("exponential", 2, "exponential", 1);
        simulateServer->simulate();

        findCalculations("exponential", 2, "exponential", 1, 0, 0, 0, 0);
        printResults();

        cout << "\n\nEXPONENTIAL UNIFORM\n";
        //EXPONENTIAL ARRIVAL ( mean = 2 sec )
        //UNIFORM SERVICE ( 1 - 2 sec )
        simulateServer = new SimulateServer("exponential", 2, "uniform", 1.2);
        simulateServer->simulate();

        findCalculations("exponential", 2, "uniform", 1.5, 0, 0, 1, 2);
        printResults();

        cout << "\n\nUNIFORM CONSTANT\n";
        //UNIFORM ARRIVAL ( 1 - 3 sec )
        //CONSTANT SERVICE ( 1 sec )
        simulateServer = new SimulateServer("uniform", 1.3, "constant", 1);
        simulateServer->simulate();

        findCalculations("uniform", 2, "constant", 1, 1, 3, 0, 0);
        printResults();

        cout << "\n\nUNIFORM EXPONENTIAL\n";
        //UNIFORM ARRIVAL ( 1 - 3 sec )
        //EXPONENTIAL SERVICE ( mean = 1 sec )
        simulateServer = new SimulateServer("uniform", 1.3, "exponential", 1);
        simulateServer->simulate();

        findCalculations("uniform", 2, "exponential", 1, 1, 3, 0, 0);
        printResults();

        cout << "\n\nUNIFORM UNIFORM\n";
        //UNIFORM ARRIVAL ( 1 - 3 sec )
        //UNIFORM SERVICE ( 1 - 2 sec )
        simulateServer = new SimulateServer("uniform", 1.3, "uniform", 1.2);
        simulateServer->simulate();

        findCalculations("uniform", 2, "uniform", 1.5, 1, 3, 1, 2);
        printResults();

        /*=======================================
        *VERIFY RESULTS
        ========================================*/

        system("PAUSE");
        return 0;
}
```