**CST-305: Project 7 – Code Errors and the Butterfly Effect**

**Objective**: Quantify the repercussions of a small code error as it propagates throughout a project.

**Description**: This assignment has two parts. In Part 1, you will construct a Lorenz System and write a computer program to visualize it. In Part 2, you will use the Lorenz System as a model to simulate and visualize the butterfly effect on code. The instructor may decide to vary the Lorenz System parameters across teams or individual implementations.

**Part 1:**

**System Performance**

This program runs a python plotting library, a python scientific mathematical package, and a python script implementing the Lorenz Function. As the program is in python, it is cross platform, and is able to run on any architecture that supports a python interpreter. The packages that the program uses are based off of C/C++ routines, giving increased performance verses a pure-python implementation.

**Specific Problem Solved:**

The efficiency of allocating memory is affected by repeated file creations and deletions. Eventually this will cause many files to be fragmented because of insufficient space and there will be a critical point where there is too much fragmentation that the system cannot operate anymore because the fragmentation causes save, load, and access times to be very slow. This phenomenon demonstrates deterministic chaos and self-organized criticality. We have gone ahead and implemented a program that uses the Lorenz system to display the fragmentation process over time. This particular system of differential equations demonstrates a file system where x is a file that has a size of 512 KBytes, y is a file that has a size of 256 KBytes, and z is a file that has a size of 512 KBytes.

**Mathematical approach for solving it:**

Systems of differential equations in real life can become extremely complex, involving hundreds or thousands of equations and variables. This vast complexity is too difficult to solve algebraically via a computer system, so computers resort to numerical methods to find solutions. One such way of using numerical methods to find solutions to systems of differential equations is through a Lorenz function. A Lorenz function goes step by step to approximate a solution.

The *r* value is the Rayleigh number, is the Prandtl number. The r value is of interest here as it governs how “far” the simulation can step forward. The greater the number, the more likely it can diverge and become chaotic.

Chaos. The level of “chaos” in a system is measured in three tiers, from lowest to highest chaos:

1. Deterministic Model. The solution can be accurately predicted and fully understood point by point.
2. Stochastic Model. The solution cannot be accurately predicted point by point, but it does converge to an equilibrium or attractor. An individual point can’t be calculated by itself, but rather is inferred by the equilibrium. To get individual points, a recursive strategy is required.
3. Chaotic Model. The solution either has no equilibrium/attractor, or has multiple equilibrium/attractors. This makes it impossible to even roughly guess where a point will be. The only way to calculate an individual point is to calculate previous solution points (recursive strategy).

This project tries to find the R value that will make the system turn from a stochastic model to a chaotic mode.

**Algorithm for code:**

1. Declare Lorenz system function.
2. Ask user for r.
3. Update the x, y, and z values step wise.
4. Graph the values on a 3D graph.
5. Graph the values of the x, y, and z each on separate 2D graphs.
6. Repeat (jump to step 2)

**Flowchart:**

**A diagram of a flowchart

Description automatically generated**

**Screenshots of the program execution**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generatedA screen shot of a graph

Description automatically generated

A screen shot of a graph

Description automatically generated

A screen shot of a graph

Description automatically generated

A screenshot of a computer screen

Description automatically generated

A screen shot of a graph

Description automatically generated

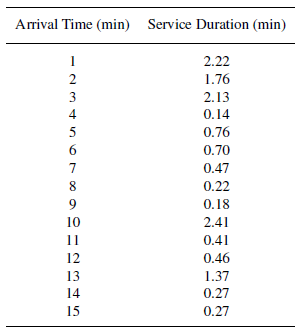
A screen shot of a graph

Description automatically generated A screen shot of a graph

Description automatically generated

**Part 2:**

1. The Table below lists the arrival times and service durations for customers in a FCFS single server queue. From this data, compute (the time average number in queue) and (the average number in queue as seen by arriving customers). For , use a time horizon of , where is the time that the last customer exits the system. Assume the system is empty at t = 0. Calculate by hand for each inter-arrival time and write a Python code and generate 5 plots, that is, (1) the customer arrival time as a function of service start time, (2) the customer arrival time as a function of exit time, (3) the customer arrival time as a function of time in queue, (4) the customer arrival time as a function of the number of customers in system and (5) the customer arrival time as a function of number of customers in queue.



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Arrival Time | Service Duration | Start Service Time | Exit Time | Time In Queue | Number In System | Number In Queue |
| 1 | 2.22 | 1.00 | 3.22 | 0 | 0 | 0 |
| 2 | 1.76 | 3.22 | 4.98 | 1.22 | 1 | 0 |
| 3 | 2.13 | 4.98 | 7.11 | 1.98 | 2 | 1 |
| 4 | 0.14 | 7.11 | 7.25 | 3.11 | 2 | 1 |
| 5 | 0.76 | 7.25 | 8.01 | 2.25 | 2 | 1 |
| 6 | 0.70 | 8.01 | 8.71 | 2.01 | 3 | 2 |
| 7 | 0.47 | 8.71 | 9.18 | 1.71 | 4 | 3 |
| 8 | 0.22 | 9.18 | 9.40 | 1.18 | 3 | 2 |
| 9 | 0.18 | 9.40 | 9.58 | 0.4 | 2 | 1 |
| 10 | 2.41 | 10.0 | 12.41 | 0 | 0 | 0 |
| 11 | 0.41 | 12.41 | 12.82 | 1.41 | 1 | 0 |
| 12 | 0.46 | 12.82 | 13.28 | 0.82 | 2 | 1 |
| 13 | 1.37 | 13.28 | 14.65 | 0.28 | 1 | 0 |
| 14 | 0.27 | 14.65 | 14.92 | 0.65 | 1 | 0 |
| 15 | 0.27 | 15.00 | 15.27 | 0 | 0 | 0 |

(1) The customer arrival time as a function of service start time

A graph on a computer screen

Description automatically generated

(2) The customer arrival time as a function of exit time

A green line graph on a white background

Description automatically generated

(3) the customer arrival time as a function of time in queue

A green line graph on a white background

Description automatically generated

(4) the customer arrival time as a function of the number of customers in system

A green line graph on a white background

Description automatically generated

(5) the customer arrival time as a function of number of customers in queue

A graph with a green line

Description automatically generated

1. On a network gateway, measurements show that the packets arrive at a mean rate of 125 packets per second and the gateway takes about 2 milliseconds to forward them. Using an M/M/1 model, analyze the gateway. What is the probability of buffer overflow if the gateway had only 12 buffers? How many buffers do we need to keep packet loss below one packet per million?

Utilization = 125/500 = 25%

The average time in entire system:

The average time waiting in line:

The average number of packets in system at any one point:

The average number of packets in a buffer at any one point: or 1/12.

This means, on average, one single buffer has a twelfth of a packet at any time. Now, with 125 packets coming in per second, this means that there must be at least 10 buffers to hold the packets coming through: 125/12 = 10.4166667, rounding to 10 buffers.

You would need 10 buffers.

This can also be verified using the utilization:

buffers needed or round up to 10 buffers.

With the machine having 12 buffers, the probability would become:

or dropping a packet less than 1 in 16 million.

1. Given an M/M/1 system (with λ < μ), suppose that we increase the arrival rate λ and the service rate μ by a factor of k each. How are the following affected?
   1. Utilization, ρ?

The utilization should not change as the *k* cancels.

A screenshot of a computer

Description automatically generated

* 1. Throughput, X?

Service time = . So, the new service time is , or the throughput is increased by a factor of k.

A screen shot of a graph

Description automatically generated

* 1. Mean number in the system, E[N]?

Now, to calculate the number of the items in the system:

The new parameters:

Now, using the answer from the next problem:

Which means that the k’s cancel.

The mean number in the system would not change. This is because the arrival rate and service rate are increased at the same rate.

A screenshot of a computer

Description automatically generated

* 1. Mean time in system, E[T]?

So, if using new parameters:

The mean time of the system will be adjusted by the factor of 1/k.

A screen shot of a graph

Description automatically generated

Show mathematically each a-d case. Using python, code and visualize your answer for each a-d.

1. In queuing theory, an M/M/1 queue refers to a queueing system with Poisson arrival process, exponential service times, and a single server. The system is defined by three parameters: λ (arrival rate), μ (service rate), and the number of servers (in this case, 1). The arrival rate can be calculated using the rules for all queues, some of which are listed below.

= The mean number of jobs in the system: , also

= The mean time a job spends in the system:

= Mean job size: , = 3 minutes

= The mean time a job spends in the queue: , < 6 minutes

= The arrival rate: = ?

= The service rate: =

= The utilization factor: =

We know that = , therefore we can substitute that into to get, We know from Little’s Law that , therefore, we can substitute in to get, . We also know that the mean time a job spends in the queue, , is equal to , and that it must be less than 6 minutes. Therefore, we can solve to find the equation for .

With this information, we can then solve for the arrival rate, in terms of .

Therefore, the mean arrival rate of jobs must be less than 2/9 or .2222222 minutes.

1. The queuing model for the Single Bus Tightly Coupled Multiprocessor (SBTCMP) architecture depicts a distributed computer system where processing elements (PEs) communicate through a shared bus and operate with finite task pools. Each PE is characterized by a CPU, a processor queue, and a bus interface unit (BIU), with independent mean service rates for the CPU and BIU. The μ(i,x) values represent the mean rates at which the operation occurs. And the p(i,x) values represent the probability that operations at that location will occur, such as branching. The model simplifies multiple BIUs into an "equivalent BIU." This queuing model offers insights into the dynamics of task execution, resource access, and communication within the SBTCMP architecture, considering factors like task migration probabilities and interrupt rates.

A paper with writing on it

Description automatically generated