Project 4 – Degradation of Data Integrity

Grant J. Burk and Ben C. Croyle

College of Science, Grand Canyon University

Professor Citro

November 3, 2024

CST-305: Benchmark – Project 5 – Self-Organized Criticality

Objective: Use a dynamical system to model the deterministically chaotic behavior of a file system.

Description:

Repeated file creations and deletions on a storage device impact the efficiency of allocating memory to store files. This results in many gaps between the files and causes a number of files to become fragmented because of insufficient contiguous space. In turn, this affects the time needed to save, load, and access a file. At some critical point, when a certain threshold of fragmentation is reached, the system becomes too slow to operate (i.e., below a predetermined threshold).

Create a model for the dynamic system that illustrates the deterministic chaos phenomenon and the self-organized criticality characteristics of a file system. Define variables for storage size, file sizes, file load time, file access time, file save time, fragmentation time, fragments assembly time, critical threshold that triggers a "system too slow" alert, and other variables as needed.

Implement the model as a computer program. The program should display a visualization of the fragmentation process as a sequence of save and delete commands are received. While you have a certain measure of freedom to choose your techniques for visualization, consult the instructor to ensure that your approach is acceptable.

Your program should also display a graph showing the appropriate metrics approaching the critical point (e.g. a line chart).

Note: *The Lab Questions in this topic provide the opportunity to reflect and experiment with mathematical and programming implementation of concepts referred to in this assignment.*

Deliverables:

1. Documentation
    a. Cover page
    b. Responsibilities and completed tasks by each team member
    c. System performance context description
    d. Specific problem solved
    e. The mathematical approach for solving it
    f. The approach for implementation in code (e.g., algorithm, flowchart)
    g. Screenshots depicting key phases in the program execution
    h. References for theory and code sources
    i. README document written in Markdown detailing how to install and run the program
2. Code
    a. Full code submitted to GitHub

   b. Code executes correctly:
      i. Reads a sequence of save and delete commands as input
      ii. Performs the commands
      iii. Saves the files as fragments if not enough contiguous space is available
      iv. Assembles files from fragments when needed
      v. Measures the overall fragmentation of the file system
      vi. Measures the time to perform tasks and the delay due to fragmentation
      vii. Creates graph showing approaching the critical point
      viii. Solution is presented using terminology and units in the context of the problem
   c. Include a header comment, including name of programmers, code packages used, and approach to implementation
   d. Include comments in key areas of the code
3. Note: Refer to the instructor for additional clarifications for the requirements of this assignment.

Responsibilities:

Code: Grant

Documentation: Ben

System performance context description
The code runs. there isn't really any loops so it should be O(n) where n is user input and the equation for time and space complexity.

Specific Problem Solved: Graphed a series of equations known as the Lorenz system in three dimensions to visualize the fragmentation of a file system due to repeated creation and deletion of files to help understand storage efficiency.

Mathematical Approach: The three equations that are collectively known as the Lorenz system are as follows: $x' = \sigma(y - x)$, $y' = rx - y - xz$, $z' = xy - bz$

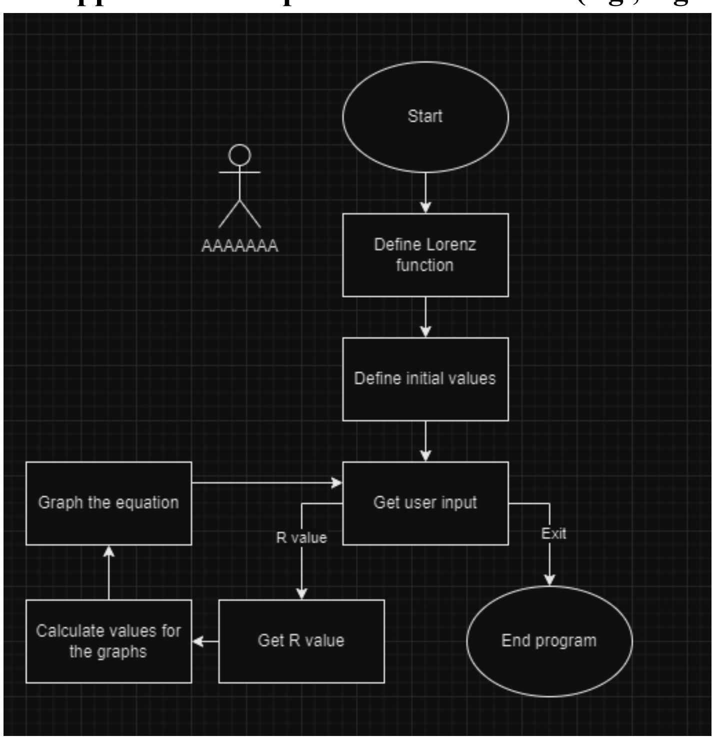
σ = 10	x = 11.8 KB	dt = 0.01
b = 2.667	y = 4.4 KB	
r = variable on user input	z = 2.4 KB	

GRAND CANYON UNIVERSITY

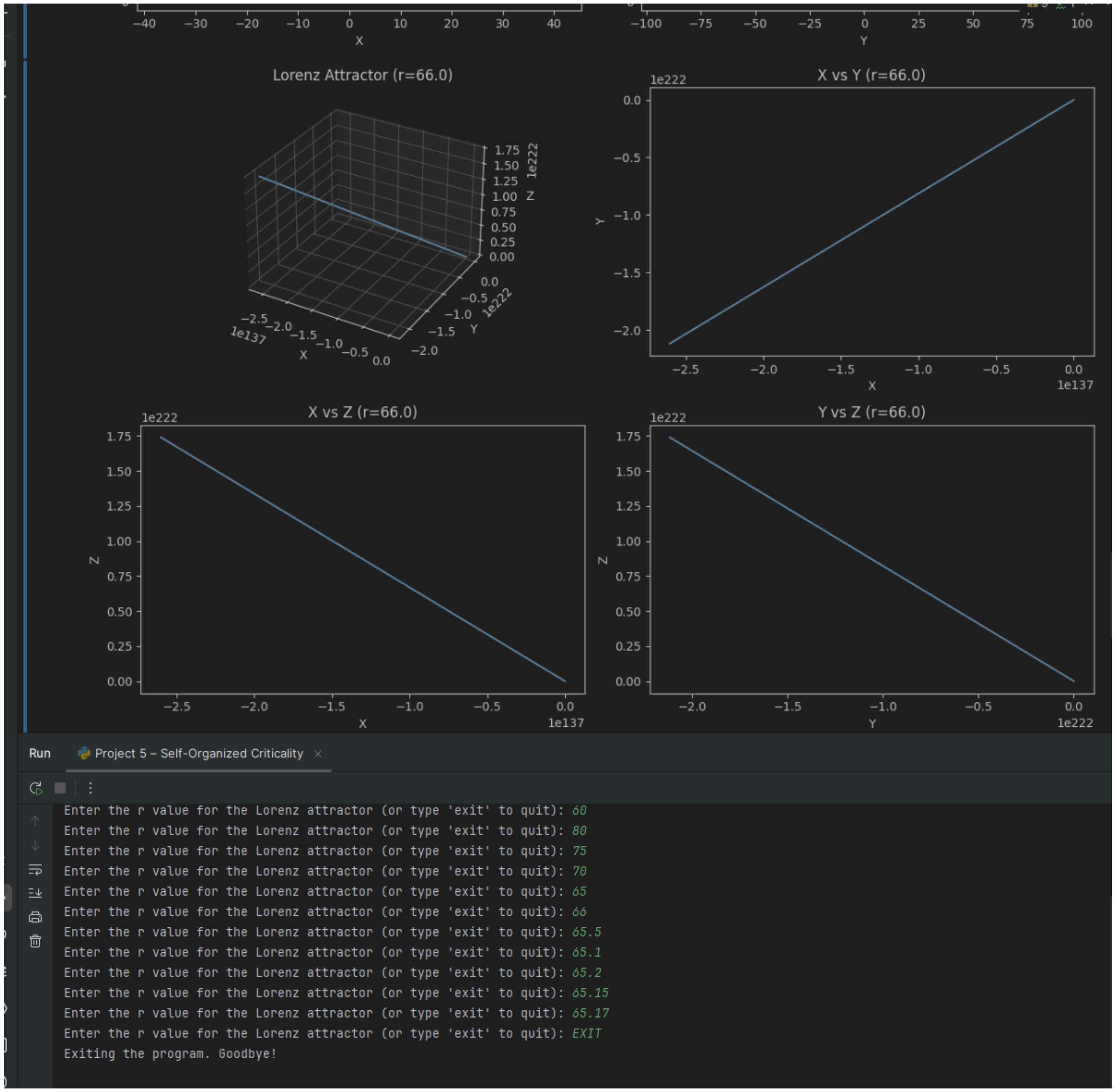
Math representation in the code:

```
def lorenz_attractor(x, y, z, dt=0.01, s=10, b=2.667, r=28):  1 usage  TaterTotMan64
    """
    Generates the Lorenz attractor.

    Args:
        x, y, z: Initial values for x, y, and z.
        dt: Time step.
        s, b, r: Lorenz parameters.

    Returns:
        x, y, z: Arrays containing the time series of x, y, and z values.
    """

    x_list, y_list, z_list = [x], [y], [z]

    for i in range(10000):
        dx = s * (y - x)
        dy = x * (r - z) - y
        dz = x * y - b * z

        x += dx * dt
        y += dy * dt
        z += dz * dt

        x_list.append(x)
        y_list.append(y)
        z_list.append(z)

    return np.array(x_list), np.array(y_list), np.array(z_list)
```

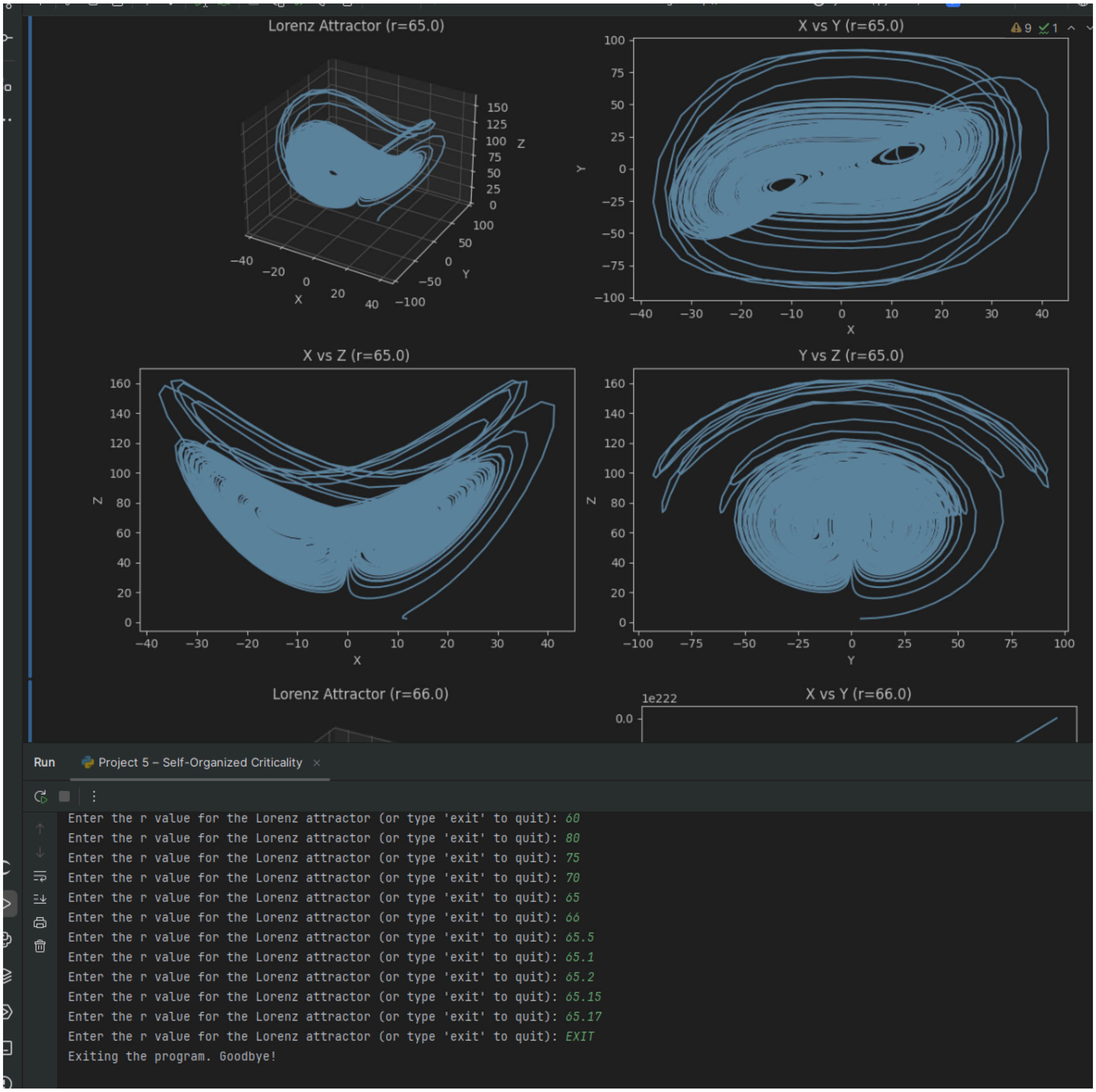
The approach for implementation in code (e.g., algorithm, flowchart)



Screenshots depicting key phases in the program execution



(Bottom shows input of r value and eventually an exit that ends the program, the top is the graphs the are outputted from the program)



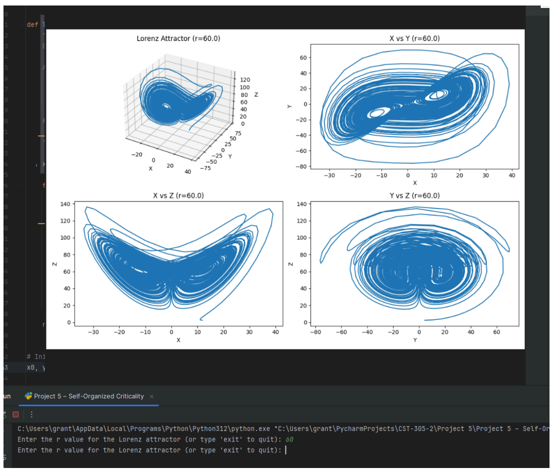
(Another diagram showing a nonlinear result)

References for theory and code sources

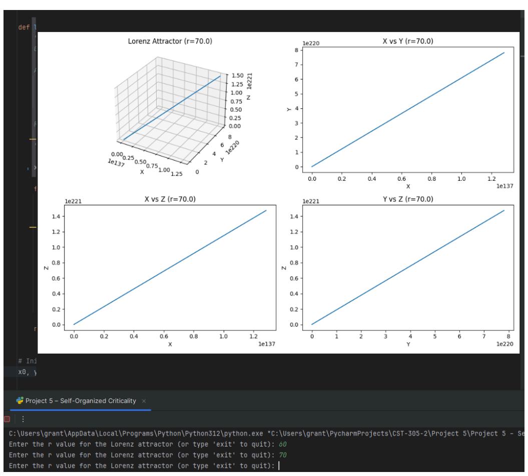
Project reference slides provided to the class.

Employing Scientific Theory + Analysis:
The goal of this experiment was to find the limit R value where the computer's storage was still usable/optimal. To identify the R value the output graphs were studied where a spiral shaped graph indicated that the system was still functioning properly while a linear graph indicates that R reached a critical point so the system doesn't work anymore as the fractionation was too extreme.

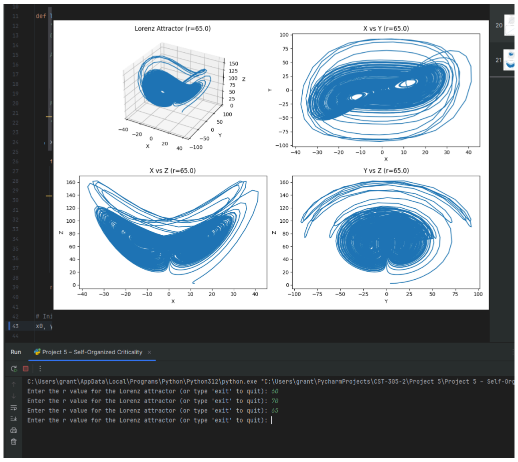
Initial test:



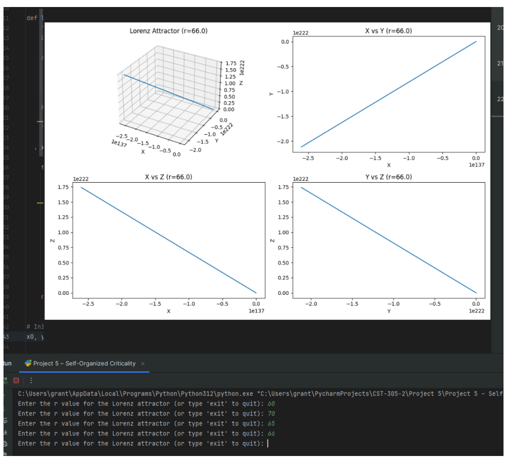
Input: 60
Analysis: as the graph is not linear, the limit is higher.



Input: 70
Analysis: the plots are linear, so the limit of the R value is lower.



Input: 65
Analysis: plots are nonlinear, so the R limit must be higher.



Input: 66
Analysis: plots are linear, so the R limit must be lower.

Conclusion:

Since 65 indicates that the limit is higher, and 66 indicates that the limit is lower, the R limit must be 65 or be somewhere between 65 and 66. 65 is the highest recorded integer R value that does not return a linear graph, so 65 is the integer limit of the system.