CST-305: Project 5- Self-Organized Criticality

Jack Utzerath

College of Technology, Grand Canyon University

CST-305: Principles of Modeling and Simulation

Ricardo Citro

11/12/23

Responsibilities and completed tasks by each team member

All tasks are done by Jack Utzerath

System performance context description

With self-organized criticality, the graphs are going to be representing different aspects of the system's behavior and state. Each graph lets the user know how file operations affect the storage system, and how it approaches criticality. The criticality is the point at which the systems become too slow due to excessive fragmentation.

Specific Problem Solved

When the system reaches the criticality point, the system becomes too slow due to excessive fragmentation. To understand this better, we can use the sandpile model. In a complex system like a file storage system, a minor event with a critical state can lead to chain reactions (avalanches) of events affecting a significant part of the system. Adding sand in this model would be the saving of files. As we continue to add sand, the pile grows until a point where adding one more grain causes a fragmentation avalanche. Redistributing sand during the avalanche would be the file fragments. The critical state of this system is sufficiently fragmented that it would slow the system down because adding another file would increase access time or require the reorganization of files. The system naturally evolves to a state where these avalanches of all sizes occur. This is otherwise known as self-organized criticality.

The mathematical approach to solving it

With the sandpile method, we can define a grid representing the storage system. Dropping and removing grains of sand is the same as adding and removing fragments. When a cell reaches a critical number of fragments, it topples and distributes its fragments to adjacent cells. We can simulate the addition and removal of files through iteration and we can keep track

of how changes propagate through the system. From this, we can observe chain reactions (avalanches).

The approach for implementation in code

My model is based on the sandpile model. In my 3D visualization, using ax.scatter, the x-axis is for time steps, y-axis is for cumulative effects, and z-axis is for the fragmentation level. This provides the user with a view of the system's evolution over time. The time steps represent the sequence of operations. The cumulative effects represent the overall usage of the storage system, and the fragmentation level indicates how fragmented the storage is at each point. The fragmentation over time plot (x-axis plot) shows the user trends regarding increasing fragmentation and the plot is good for identifying the point at which the system might reach a critical state. The fragmentation vs cumulative effects plot (y-axis plot) can help the user identify if there's a direct correlation between the number of operations and the level of fragmentation. The fragmentation distribution plot (z-axis plot) can highlight areas of high fragmentation where file operations are concentrated. Each bar represents a position in the storage, and the height of the bar indicates the fragmentation level at that position.

References for theory and code sources

Abelian Sandpile Model#. Abelian Sandpile Model - Thematic Tutorials. (n.d.).

https://doc.sagemath.org/html/en/thematic_tutorials/sandpile.html

Readme Document written in Markdown detailing how to install and run the program

Full code submitted to GitHub

Github SC

Code:

Next Page

```python
#Jack Utzerath
#Self Organized Criticality

# Import necessary libraries for simulation and plotting.
import matplotlib.pyplot as plt
import numpy as np
import random
import time

# Initialize variables for the simulation.
# storage_size simulates the total capacity of the file storage system.
storage_size = 1000
# C_threshold represents the fragmentation threshold beyond which the system is too slow.
C_threshold = 4
# Initialize the storage structure, representing the state of each storage unit.
storage = [0 for _ in range(storage_size)]

# Define r values that will simulate different rates of file operations.
r_values = [0.2, 0.5, 0.8]

# Function to simulate the addition of a file in the storage system.
def add_file(storage, position, size):
    """
    Simulates adding a file to the storage, increasing the fragmentation.
    Triggers an avalanche if the fragmentation threshold is exceeded.
    """
    if position + size <= len(storage):
        for i in range(position, position + size):
            storage[i] += 1
            if storage[i] >= C_threshold:
                trigger_fragmentation_avalanche(storage, i)

# Function to simulate a fragmentation avalanche.
def trigger_fragmentation_avalanche(storage, position):
    """Distribute fragments in a sandpile-like manner."""
    if 0 <= position < len(storage):
        if storage[position] >= C_threshold:
            storage[position] -= C_threshold
            if position > 0:
                storage[position - 1] += 1
                # Recursively trigger avalanches in neighboring cells
                if storage[position - 1] >= C_threshold:
                    trigger_fragmentation_avalanche(storage, position - 1)
            if position + 1 < len(storage):
                storage[position + 1] += 1
                # Recursively trigger avalanches in neighboring cells
                if storage[position + 1] >= C_threshold:
                    trigger_fragmentation_avalanche(storage, position + 1)
```

```python
# Function to get the current simulation time.
def current_simulation_time():
    """
    Returns the current wall-clock time to simulate file operation times.
    """
    return time.time()

# Simulation function that uses the r values to simulate the file system over time.
def run_simulation(r, storage_size, max_iterations=1000):
    """
    Runs the simulation for a given rate of file operations (r value).
    Tracks the time taken for file operations and the state of the storage system.
    """
    storage = [0 for _ in range(storage_size)]
    time_steps = []
    cumulative_effects = []

    # Initialize variables to track the time of operations.
    file_save_time = 0
    file_load_time = 0
    fragmentation_time = 0
    fragments_assembly_time = 0

    # Simulation loop for file operations.
    for time_step in range(max_iterations):
        file_size = int(r * storage_size)
        position = random.randint(0, storage_size - file_size)

        # Timing the save operation.
        start_time = current_simulation_time()
        add_file(storage, position, file_size)
        file_save_time += current_simulation_time() - start_time

        # Tracking the cumulative effects of fragmentation.
        time_steps.append(time_step)
        cumulative_effects.append(sum(storage))

    # Ensure all arrays have the same length for plotting.
    if len(time_steps) != len(cumulative_effects) or len(cumulative_effects) != len(storage):
        min_length = min(len(time_steps), len(cumulative_effects), len(storage))
        time_steps = time_steps[:min_length]
        cumulative_effects = cumulative_effects[:min_length]
        storage = storage[:min_length]

    return time_steps, cumulative_effects, storage, file_save_time, file_load_time, fragmentation_time, fragments_assembly_time
```

```python
# Run the simulation for each r value and collect results.
results = {}
from mpl_toolkits.mplot3d import Axes3D
for r in r_values:
    results[r] = run_simulation(r, storage_size)

# Plotting the results in 3D as well as on individual axes to visualize the self-organized criticality.
for r in r_values:
    # Extract data from the simulation results.
    time_steps, cumulative_effects, storage, _, _, _, _ = results[r]

    # 3D Plot to visualize the state of the system over time.
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.scatter(time_steps, cumulative_effects, storage)
    ax.set_title(f"3D Visualization for r={r}")
    ax.set_xlabel('Time Step (X-axis)')
    ax.set_ylabel('Cumulative Effects (Y-axis)')
    ax.set_zlabel('Fragmentation Level (Z-axis)')
    plt.show()



    # X-axis plot
    plt.plot(time_steps, storage)
    plt.title(f"Fragmentation Over Time for r={r}    (X-axis plot)")
    plt.xlabel('Time Step')
    plt.ylabel('Fragmentation Level')
    plt.show()

    # Y-axis plot
    plt.plot(cumulative_effects, storage)
    plt.title(f"Fragmentation vs Cumulative Effects for r={r}   (y-axis plot)")
    plt.xlabel('Cumulative Effects')
    plt.ylabel('Fragmentation Level')
    plt.show()

    # Z-axis plot
    plt.bar(range(len(storage)), storage)
    plt.title(f"Fragmentation Distribution for r={r}    (z-axis plot)")
    plt.xlabel('Storage Position')
    plt.ylabel('Fragmentation Level')
    plt.show()
```

Code Output

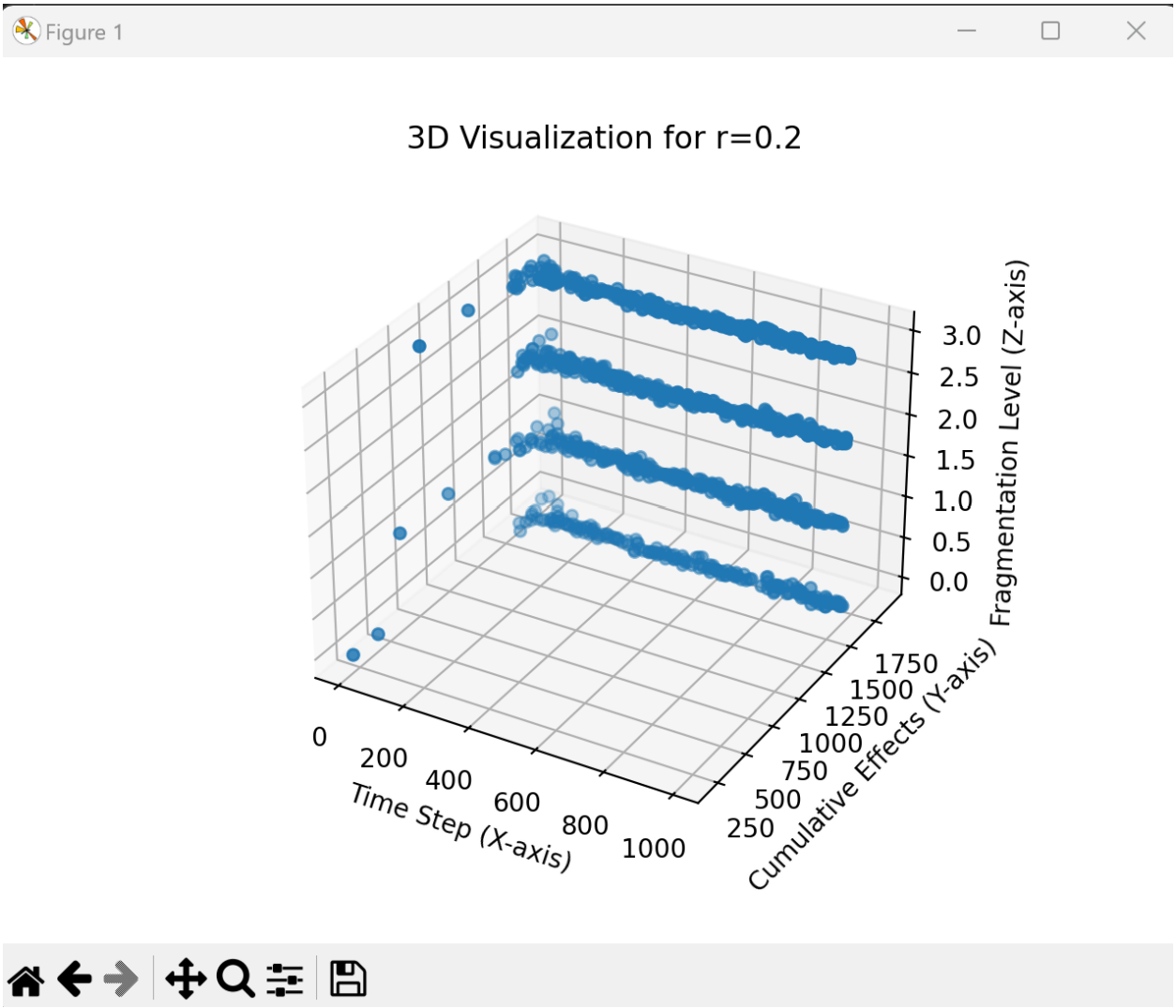
Figure 1
3D Visualization for r=0.2
Fragmentation Level (Z-axis)
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Cumulative Effects (Y-axis)
1750
1500
1250
1000
750
500
250
Time Step (X-axis)
0
200
400
600
800
1000

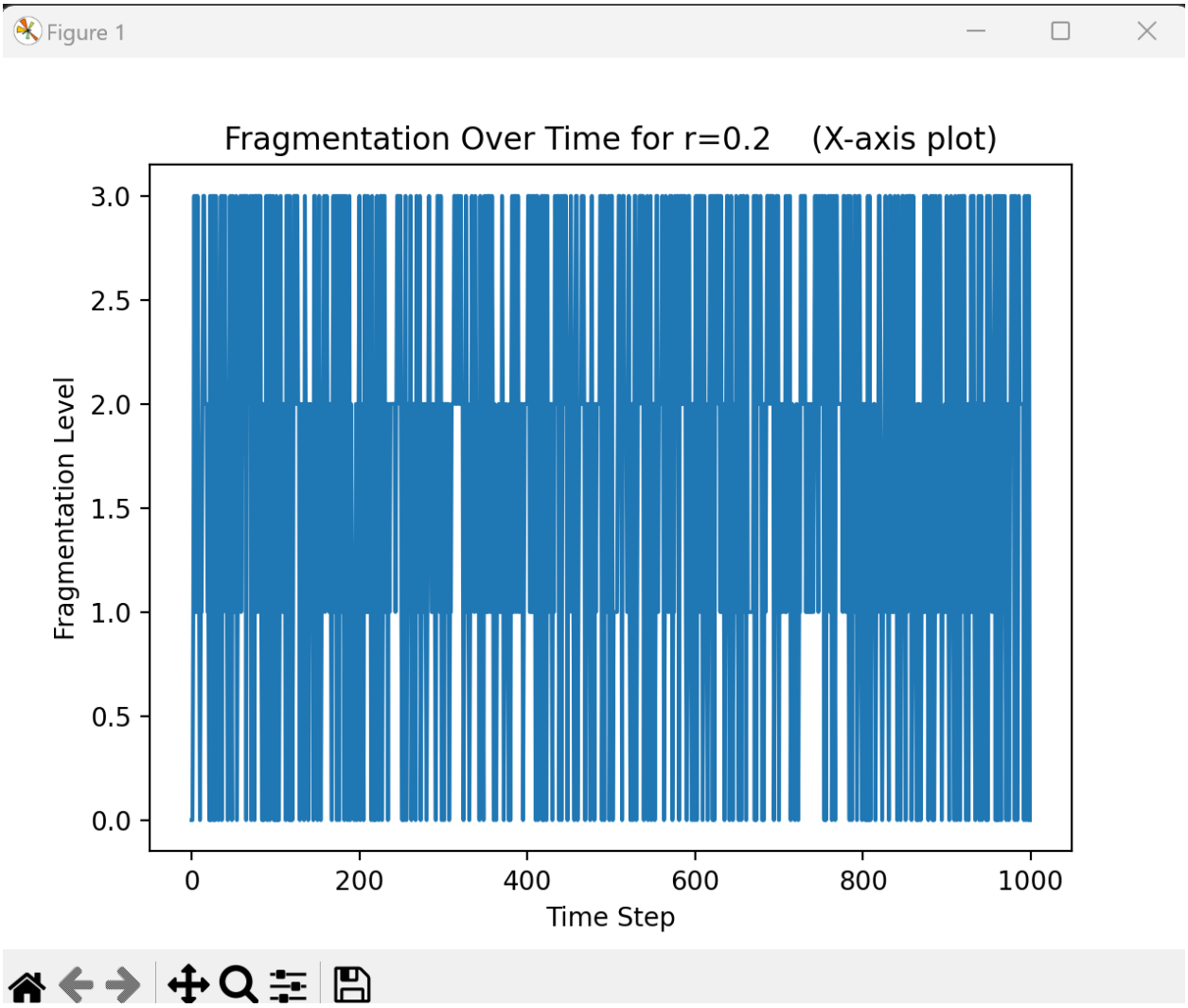
Figure 1
Fragmentation Over Time for r=0.2 (X-axis plot)
Fragmentation Level
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
200
400
600
800
1000
Time Step

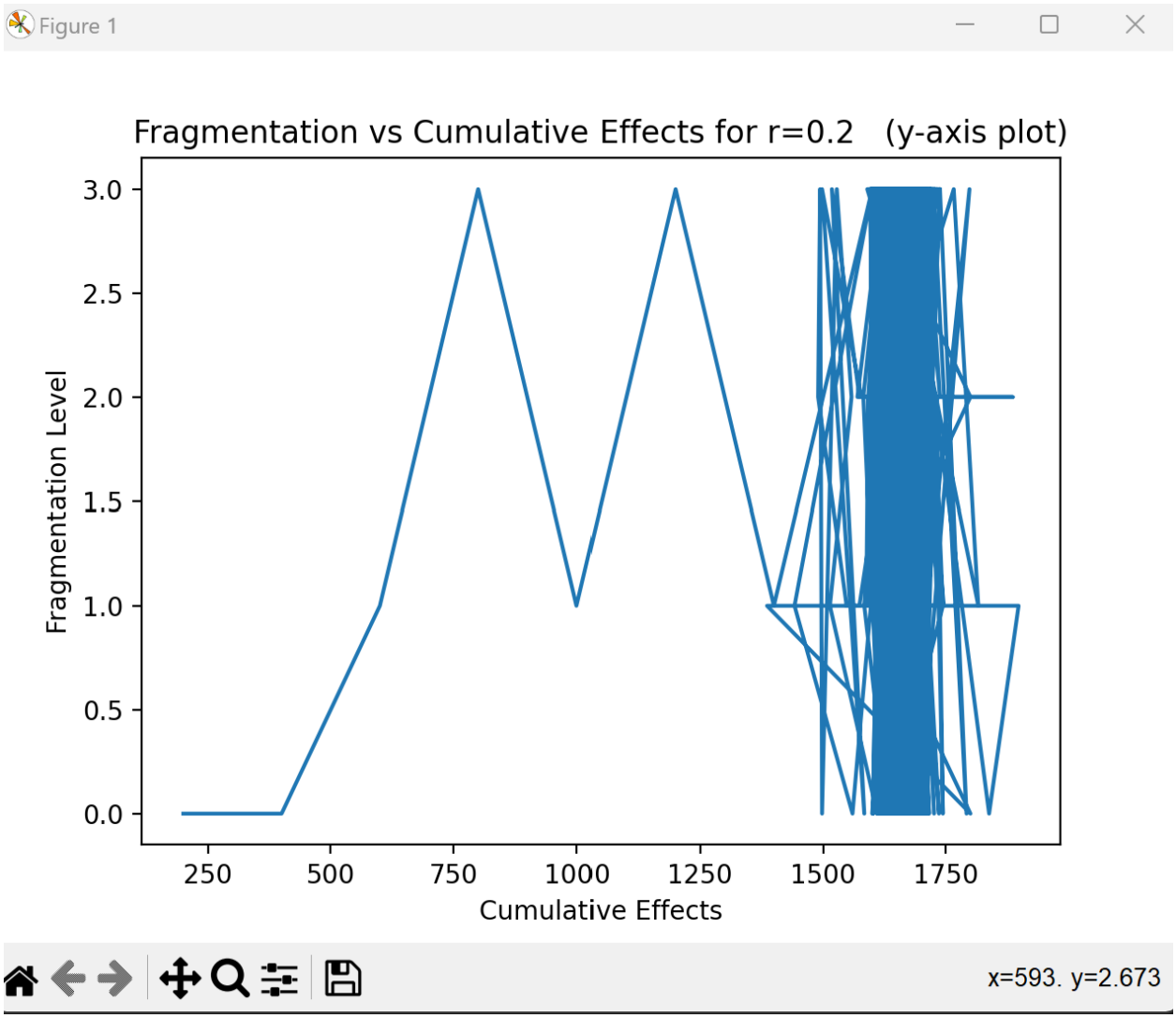
Figure 1
Fragmentation vs Cumulative Effects for r=0.2 (y-axis plot)
Fragmentation Level
Cumulative Effects
3.0
2.5
2.0
1.5
1.0
0.5
0.0
250
500
750
1000
1250
1500
1750
x=593. y=2.673

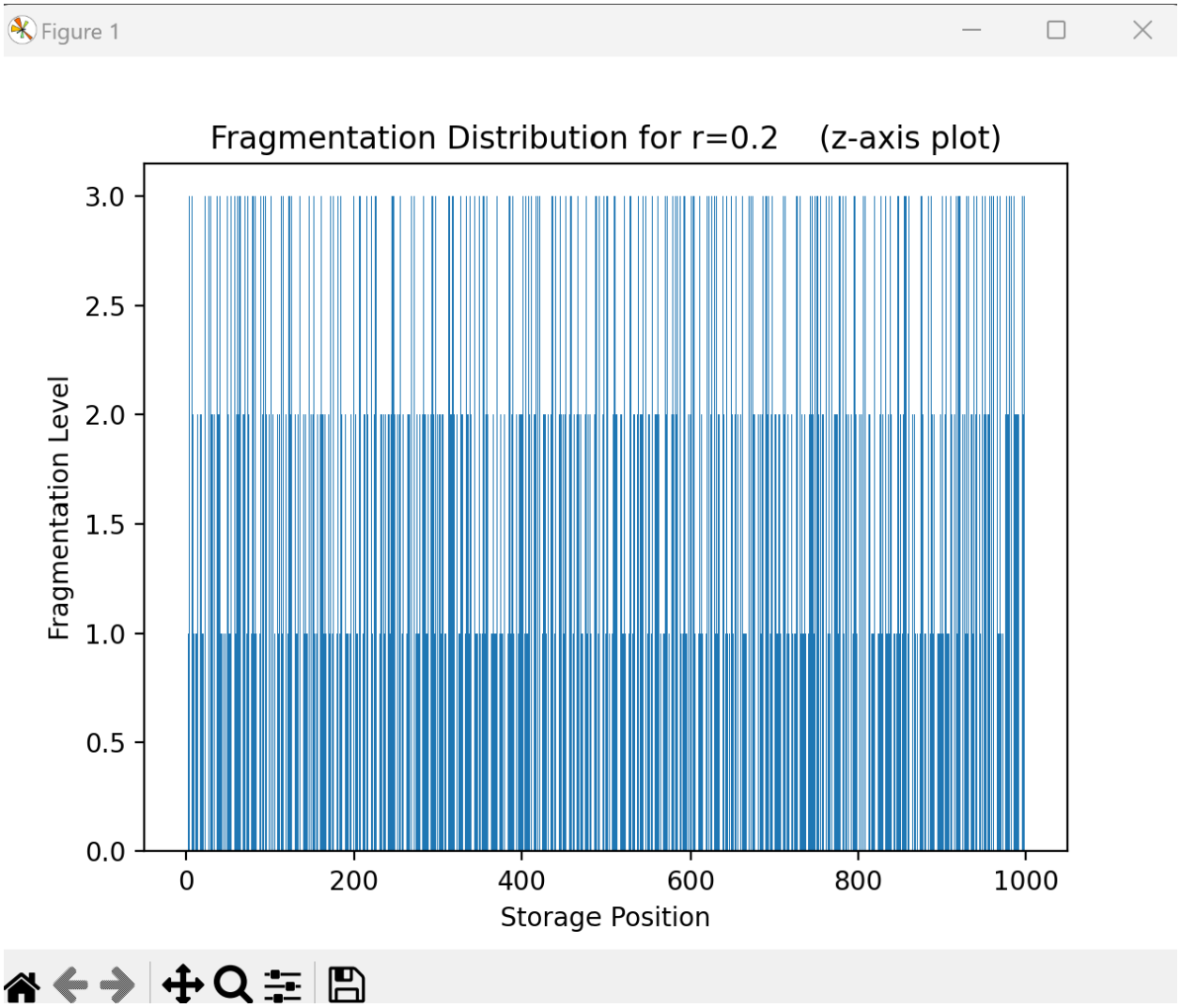
Figure 1
Fragmentation Distribution for r=0.2 (z-axis plot)
Fragmentation Level
Storage Position
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
200
400
600
800
1000

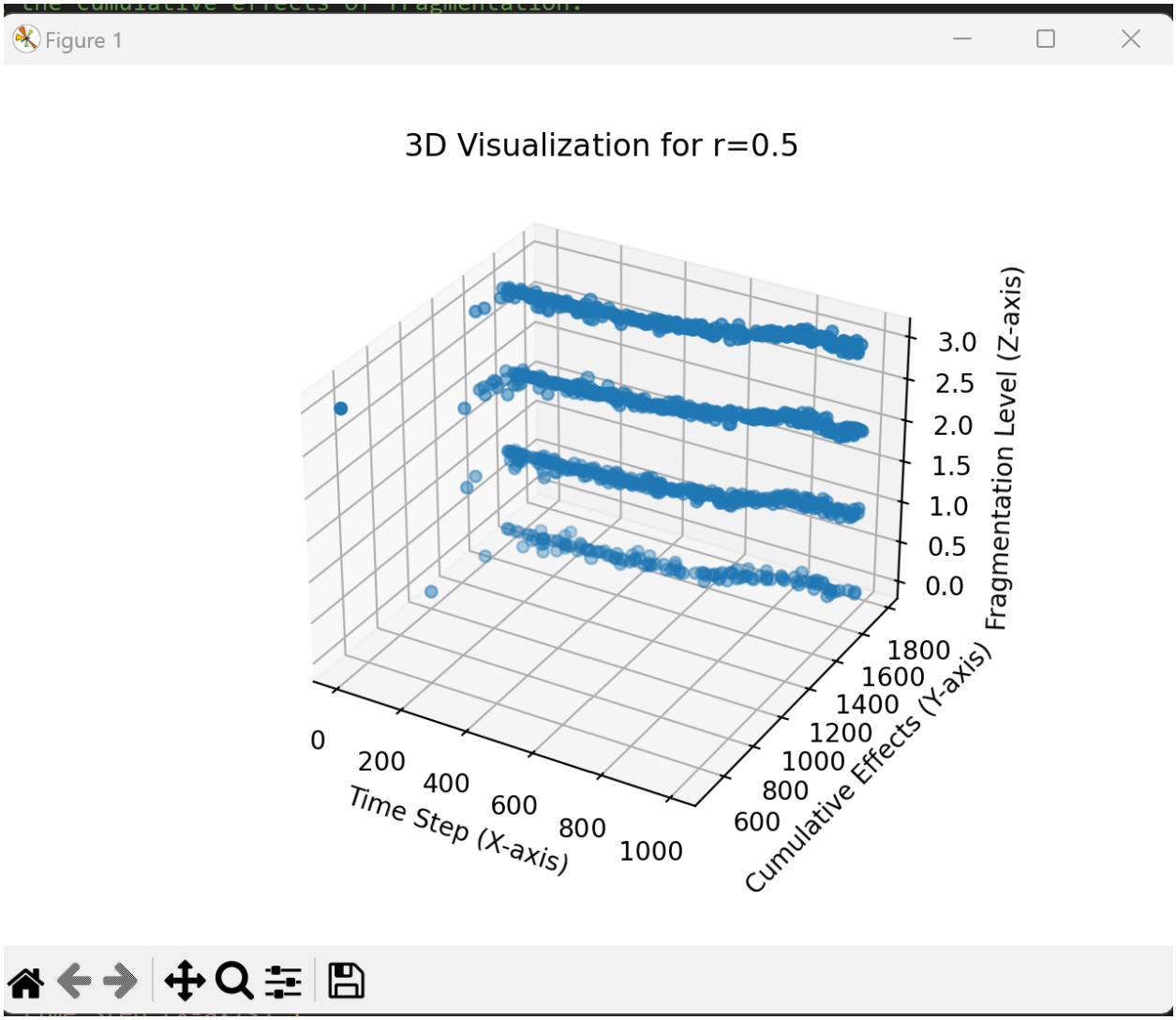
Figure 1
3D Visualization for r=0.5
Time Step (X-axis)
0
200
400
600
800
1000
Cumulative Effects (Y-axis)
600
800
1000
1200
1400
1600
1800
Fragmentation Level (Z-axis)
0.0
0.5
1.0
1.5
2.0
2.5
3.0

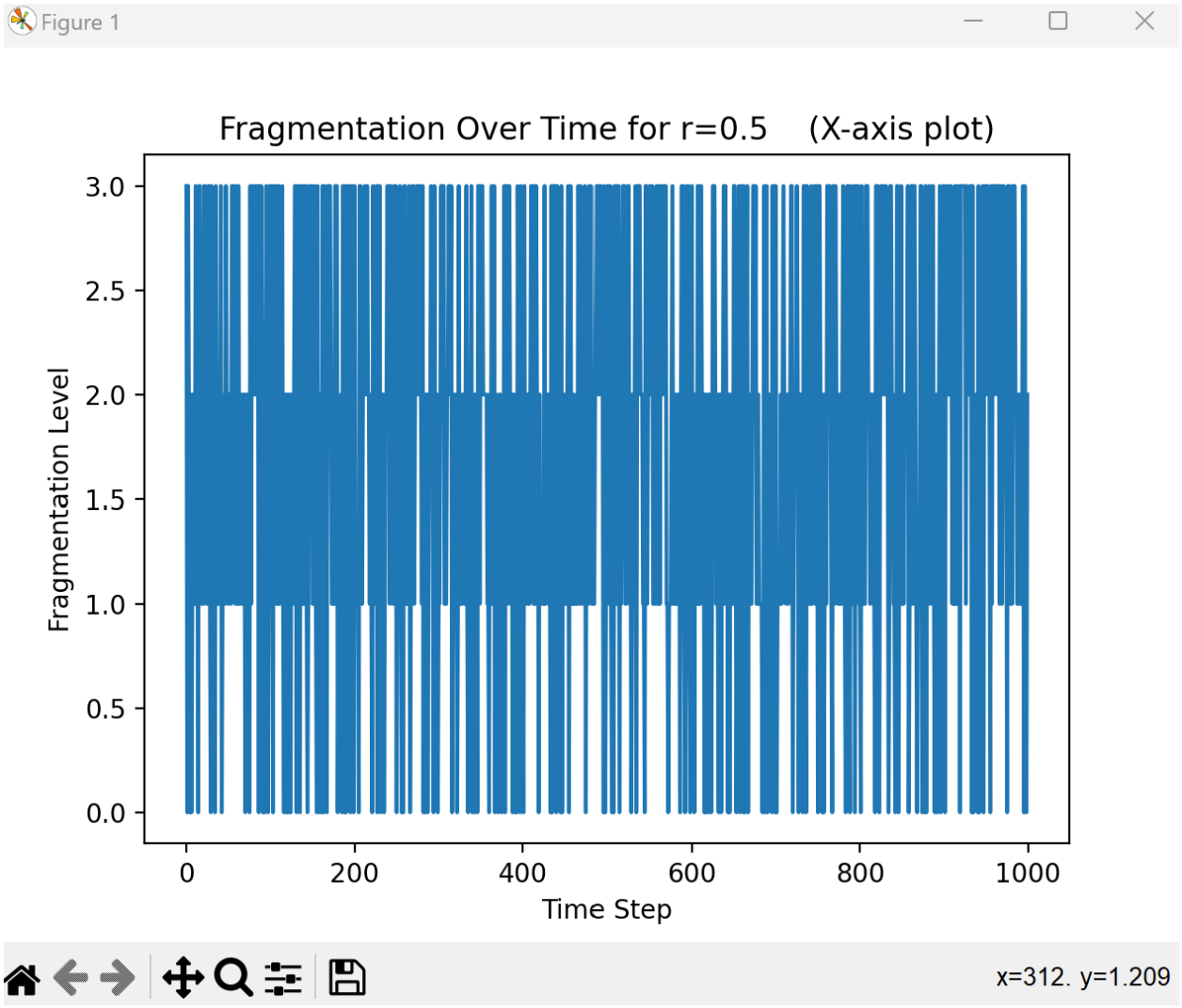
Figure 1
Fragmentation Over Time for r=0.5 (X-axis plot)
Fragmentation Level
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
200
400
600
800
1000
Time Step
x=312. y=1.209

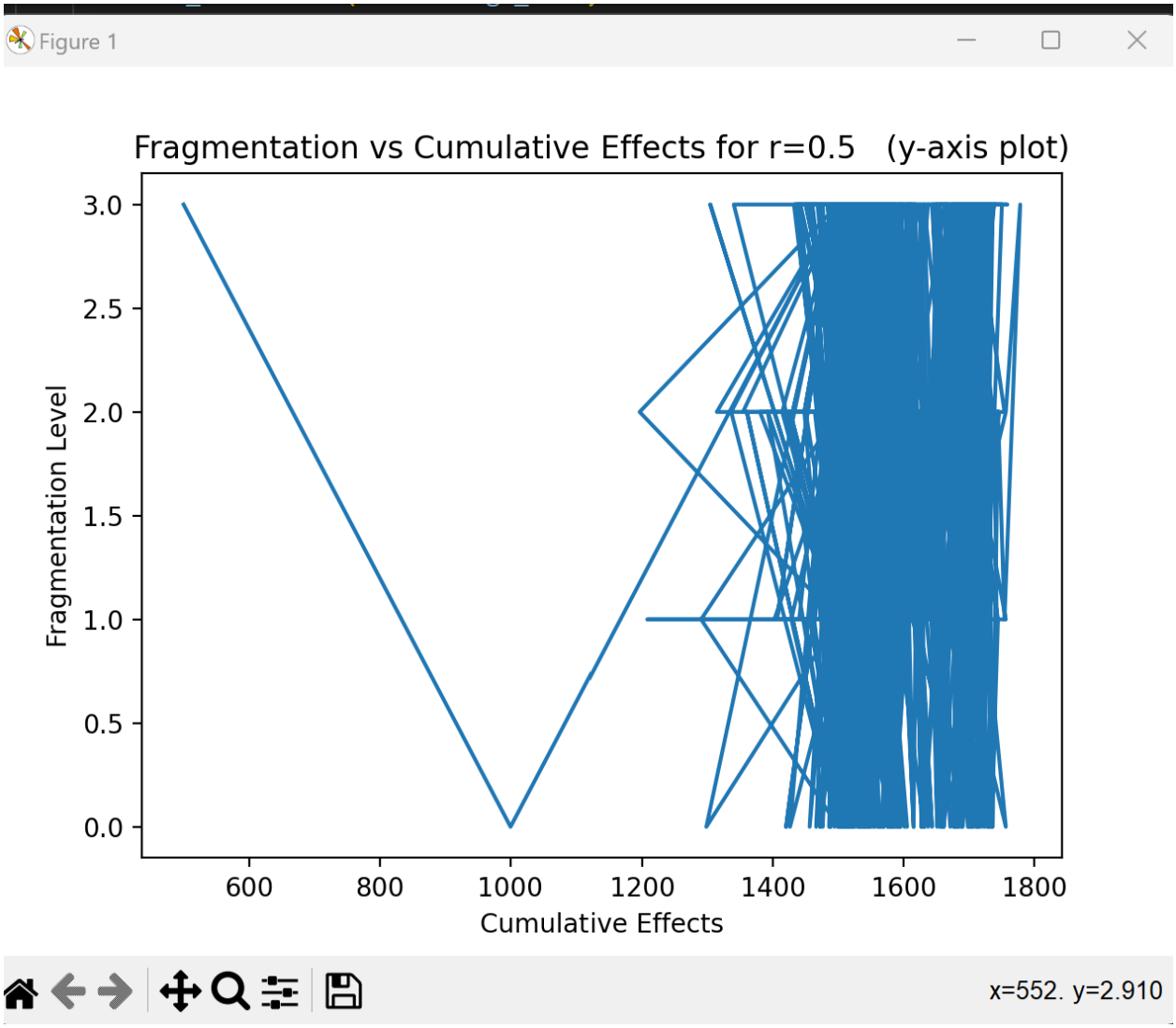
Figure 1
Fragmentation vs Cumulative Effects for r=0.5 (y-axis plot)
Fragmentation Level
3.0
2.5
2.0
1.5
1.0
0.5
0.0
600
800
1000
1200
1400
1600
1800
Cumulative Effects
x=552. y=2.910

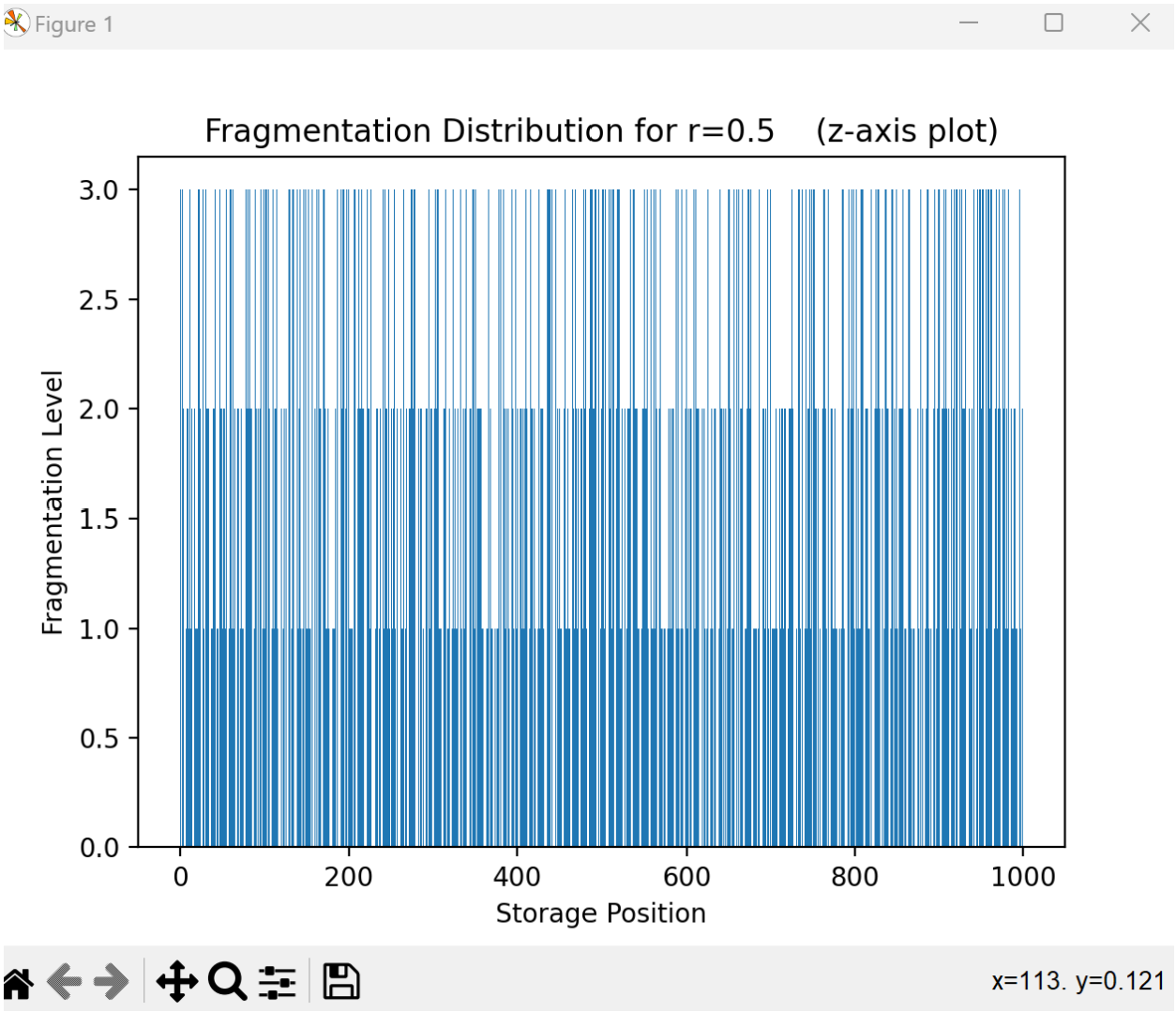
Figure 1
Fragmentation Distribution for r=0.5 (z-axis plot)
Fragmentation Level
Storage Position
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
200
400
600
800
1000
x=113. y=0.121

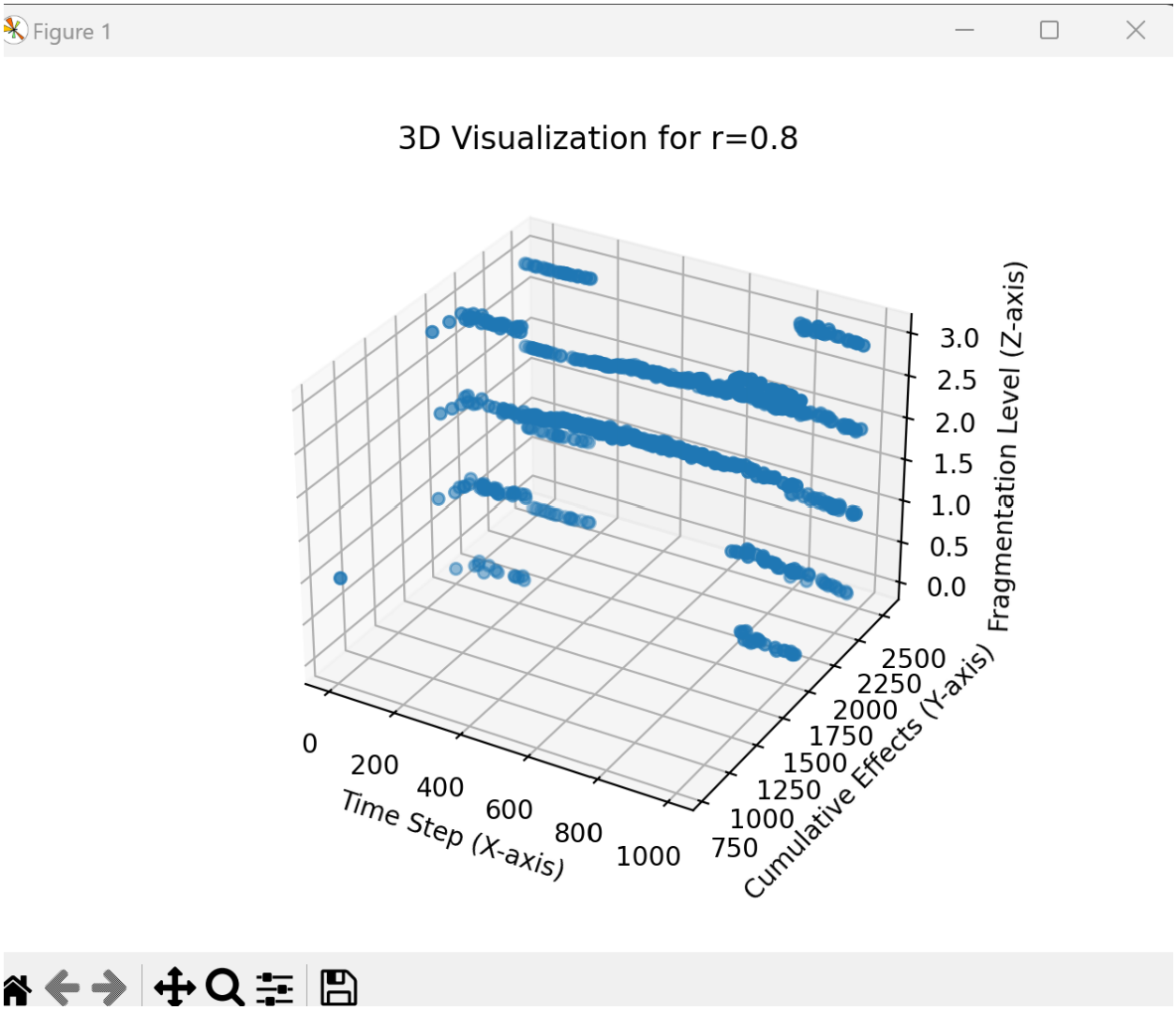
Figure 1
3D Visualization for r=0.8
Fragmentation Level (Z-axis)
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Cumulative Effects (Y-axis)
2500
2250
2000
1750
1500
1250
1000
750
Time Step (X-axis)
0
200
400
600
800
1000

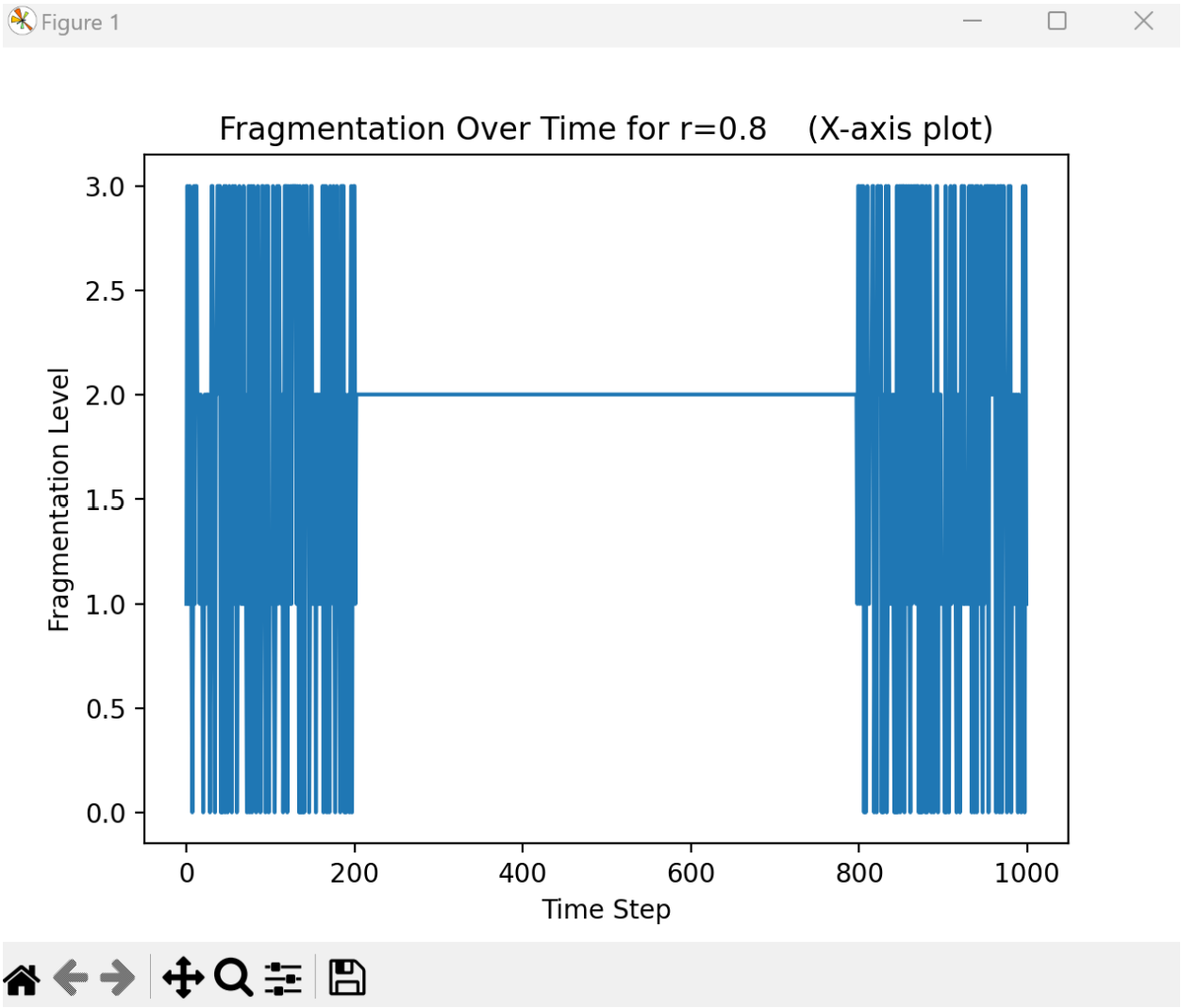
Figure 1
Fragmentation Over Time for r=0.8 (X-axis plot)
Fragmentation Level
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
200
400
600
800
1000
Time Step

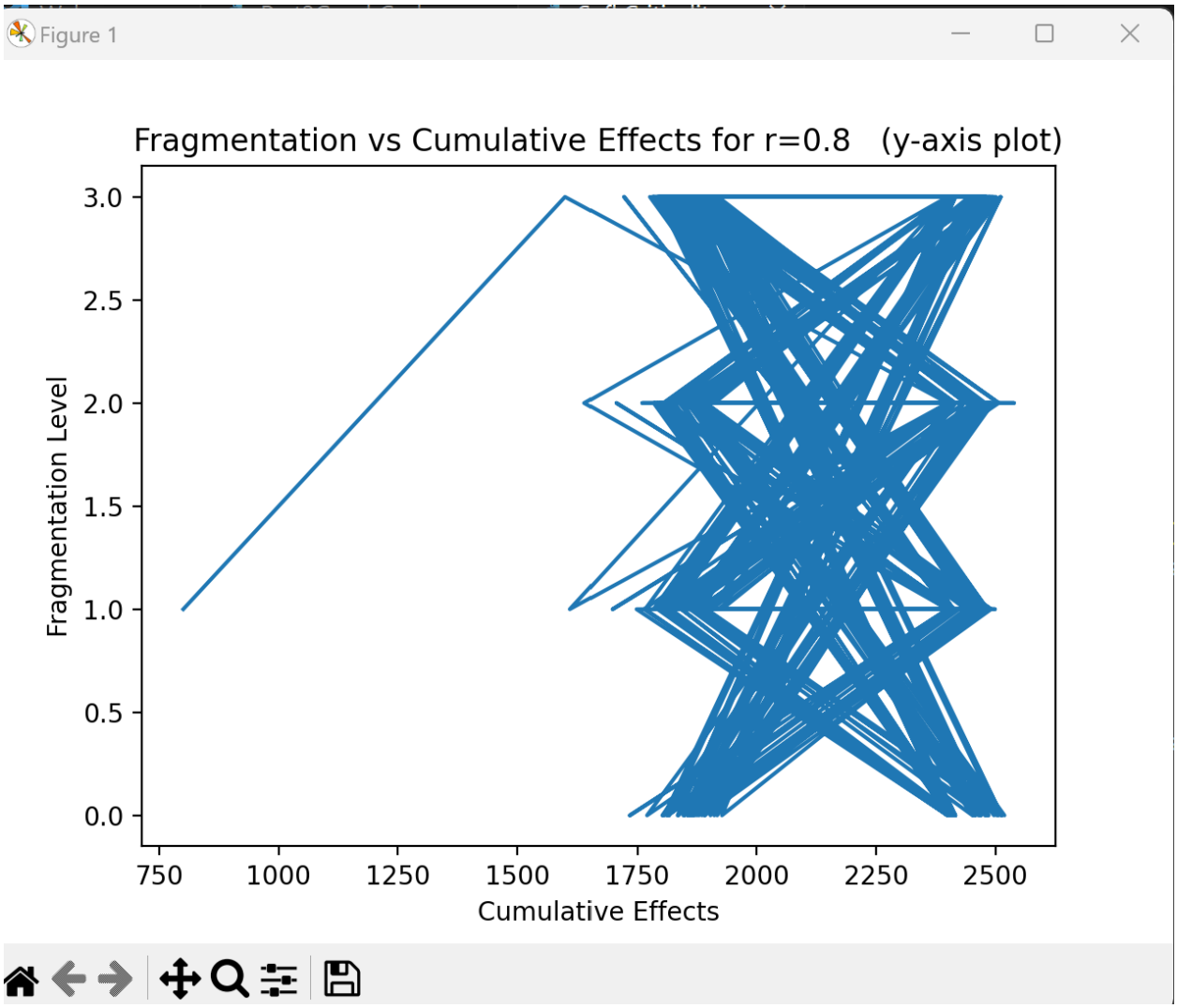
Figure 1
Fragmentation vs Cumulative Effects for r=0.8 (y-axis plot)
Fragmentation Level
3.0
2.5
2.0
1.5
1.0
0.5
0.0
750
1000
1250
1500
1750
2000
2250
2500
Cumulative Effects

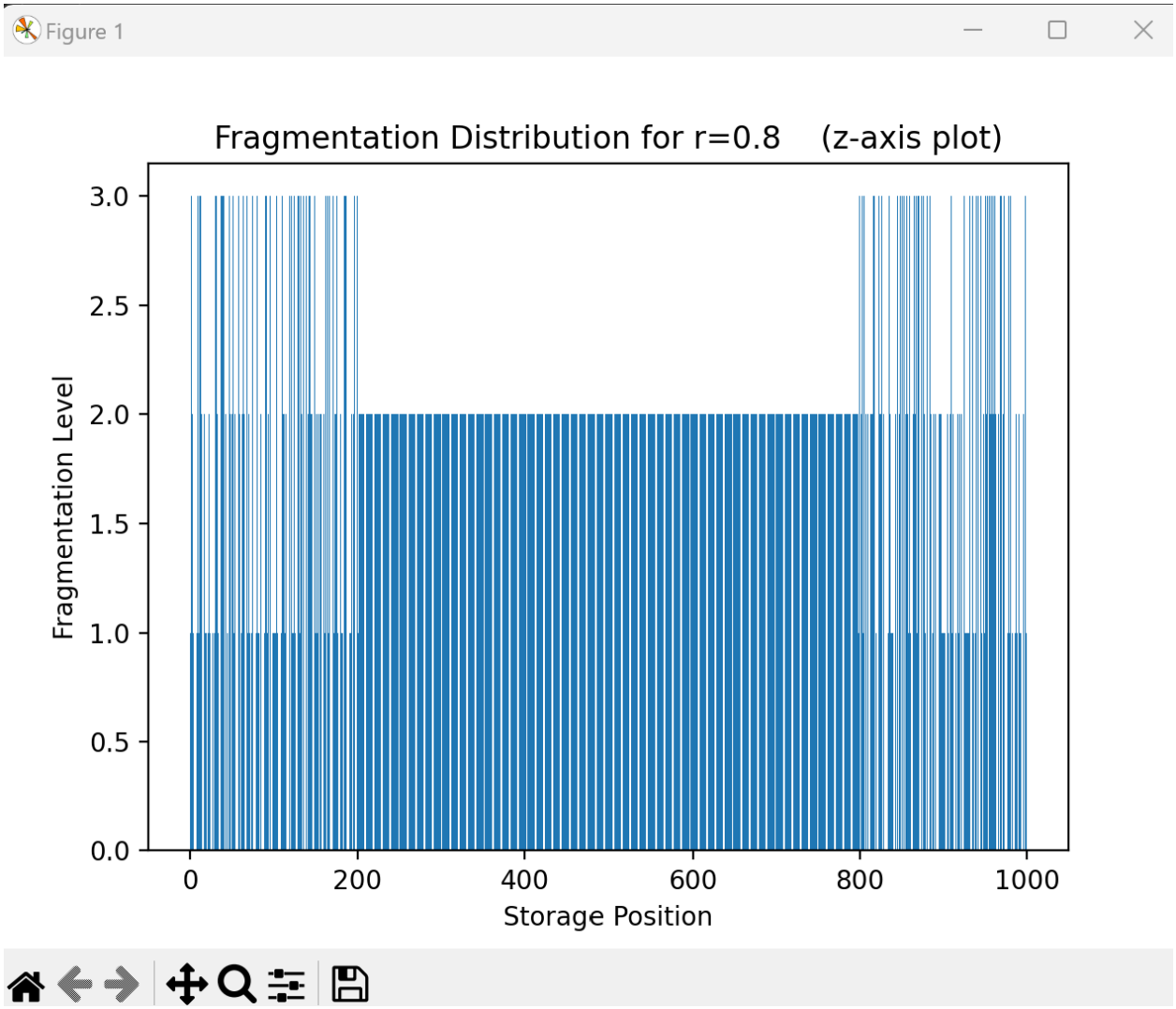
Figure 1
Fragmentation Distribution for r=0.8 (z-axis plot)
Fragmentation Level
Storage Position
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
200
400
600
800
1000