Complex Systems

CS2024/problem_3.pdf

Result

3 Self-organized criticality: the Oslo model

The Oslo rice-pile model is a theoretical model of self-organized criticality (SOC) used to study the behavior of granular materials and avalanche-like phenomena in a simple, discrete system. It was introduced as a variation of the sandpile model, and it's named after the city of Oslo, Norway, where it was developed.

1. Implement the Oslo model using the following algorithm focusing on slopes z_i :
   - (a) Initialize the system in arbitrary stable configuration $z_i \leqslant z_t^T$, where z_i^T is i-th slope threshold $\in \{1, 2\}$;
   - (b) Drive the system by adding a grain at left-most site;
   - (c) If $z_i < z_i^T$, relax the site i,

     for $i = 1 : z_1 \to z_1 - 2, z_2 \to z_2 + 1$;

     for $i = 2 \ldots L-1 : z_i = z_i - 2, z_{i\pm1} \to z_{i\pm1} + 1$;

     for $i = L : z_L \to z_L - 1, z_{L-1} \to z_{L-1} + 1$;

     During relaxation do not forget about choosing randomly new threshold $z_t^T i \in \{1, 2\}$ for the relaxed site. Continue relaxation until $z_i \leqslant z_i^T$ for all i;
   - (d) Return to point (b).

Listing 1: Oslo Model Algorithm (Python version 3.11.7)

```python
import numpy as np
import matplotlib.pyplot as plt

def oslo_model(L, T, z_thresholds=[1, 2]):
    """
    Simulates the Oslo model of self-organized criticality.

    Parameters:
    - L: int, size of the system (number of sites).
    - T: int, number of grain additions (time steps).
    - z_thresholds: list, possible slope thresholds (default is [1, 2])
        .

    Returns:
    - avalanches: list, size of avalanches during the simulation.
    """

    slopes = np.random.choice(z_thresholds, size=L)  # (a) Initial
        random slopes in {1, 2}
    thresholds = np.random.choice(z_thresholds, size=L) # (a) Initial
        random thresholds in {1, 2}
    avalanches = []

    for t in range(T):
        slopes[0] += 1 # (b) Add a grain to the left-most site
        avalanche_size = 0

```

```python
        # (c) Relaxation process
        while np.any(slopes > thresholds):
            for i in range(L):
                if slopes[i] > thresholds[i]:
                    avalanche_size += 1
                    thresholds[i] = np.random.choice(z_thresholds)  # Assign a new threshold

                    if i == 0:  # Left-most site
                        slopes[i] -= 2
                        slopes[i + 1] += 1
                    elif i == L - 1:  # Right-most site
                        slopes[i] -= 1
                        slopes[i - 1] += 1
                    else:  # Internal sites
                        slopes[i] -= 2
                        slopes[i - 1] += 1
                        slopes[i + 1] += 1

        avalanches.append(avalanche_size)

    return avalanches
```

2. Plot scaled avalanche size s/s_{max} in function of time t (measured in terms of grain additions). Does it make sense to analyze data for small t? Which condition should be satisfied in avalanche size statistical analysis?

Listing 2: Oslo Model Algorithm (Python version 3.11.7)

```python
L = 2  # System length
T = 4  # Number of time steps
avalanches = oslo_model(L, T)

# Scaled avalanche sizes
s_max = max(avalanches)
scaled_avalanches = [s / s_max for s in avalanches]

# Plotting
plt.figure(figsize=(10, 6))
plt.plot(range(T), scaled_avalanches, label='Scaled Avalanche Size')
plt.xlabel('Time (Grain Additions)')
plt.ylabel('Scaled Avalanche Size $s/s_{max}$')
plt.title('Scaled Avalanche Size vs. Time')
plt.grid(True)
plt.legend()
plt.savefig('avalanche_one.png')
plt.show()
```

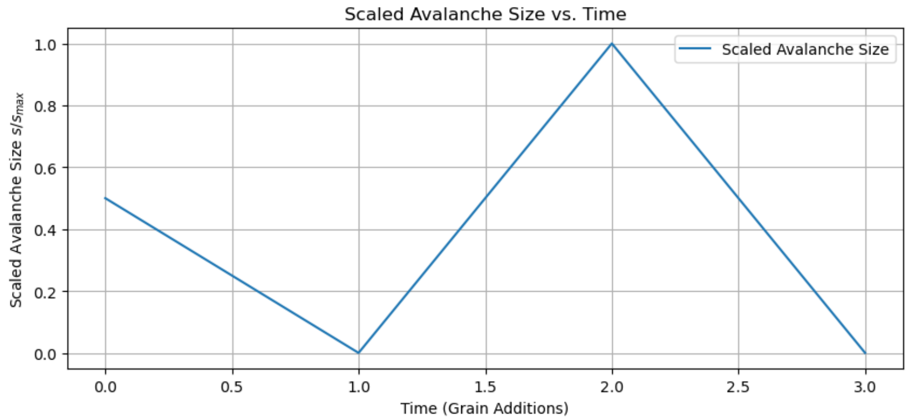

Figure 1: Plot for L = 2, T = 4

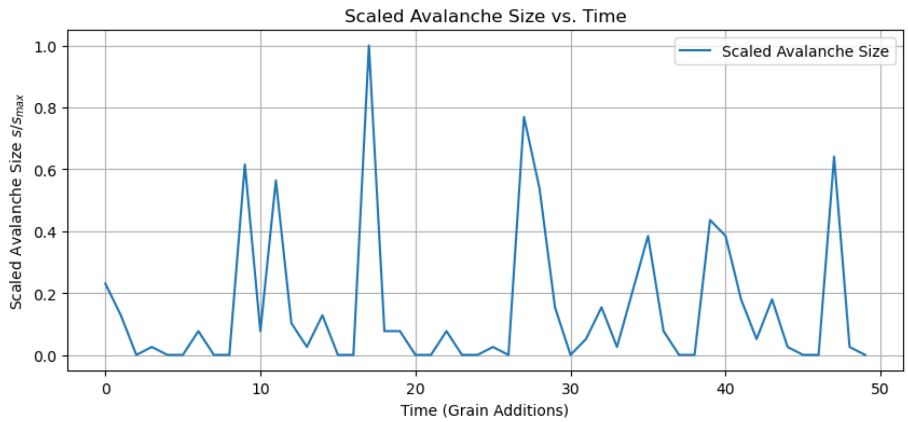

Figure 2: Plot for L = 6, T = 50

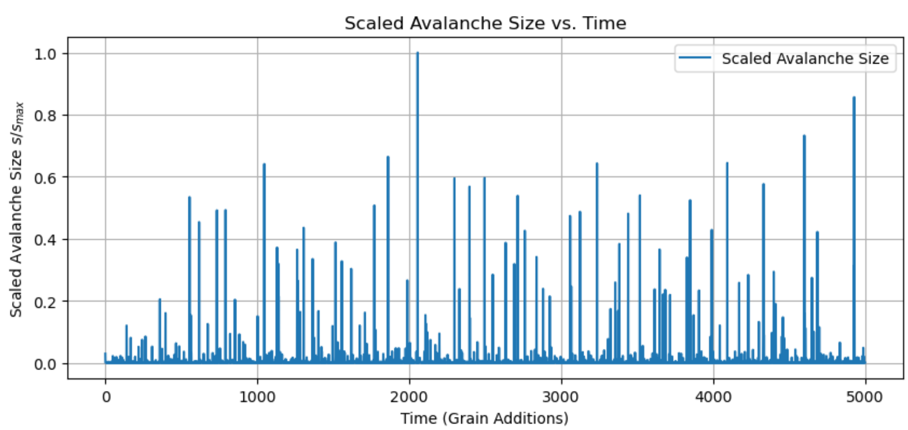

Figure 3: Plot for L = 64, T = 5000

Does it makes sense to analyze data for small t?

ANSWER: It doesn't make sense to analyze data for small t because the system has not reached its critical state yet, and the results will not be representative of the true behavior of the system.

What condition should be satisfied in avalanche size statistical analysis?

ANSWER: The system should be in steady state, and you need a large number of grains added to the system for meaningful results (to observe the power-law distribution of avalanche sizes).

3. Plot in log-log scale avalanche size probability $P(s, L)$ with respect to avalanche size s for several lengths of the system (L should be at least 64). Do you observe power law behavior? Why does this power law break for large s?

Listing 3: Oslo Model Algorithm (Python version 3.11.7)

```python
from scipy.stats import linregress
import numpy as np
import matplotlib.pyplot as plt


def avalanche_probability(avalanches, L_values, name, min_s=1, max_s=200):
    """
    Plot avalanche size probability P(s, L) in log-log scale with points
        and linear regression.

    Parameters:
    - avalanches: list of lists, avalanches for different system lengths.
    - L_values: list of int, system lengths.
    - name: str, name for the saved plot file.
    - min_s: float, minimum avalanche size for fitting the power-law.
    - max_s: float, maximum avalanche size for fitting the power-law.
    """
    plt.figure(figsize=(10, height))

    for avals, L in zip(avalanches, L_values):
        if len(avals) == 0:
            print(f"No data for L={L}, skipping.")
            continue

        # Remove non-positive values
        avals = np.array([a for a in avals if a > 0])

        if len(avals) == 0:
            print(f"All data for L={L} are zero or invalid, skipping.")
            continue

        # Calculate probabilities
        avalanche_sizes, counts = np.unique(avals, return_counts=True)
        probabilities = counts / sum(counts)

        # Plot points
```

```python
        plt.scatter(avalanche_sizes, probabilities, label=f'L={L}', alpha=0.7)

        # Filter for power-law range
        valid_range = (avalanche_sizes >= min_s) & (avalanche_sizes <= max_s)
        filtered_sizes = avalanche_sizes[valid_range]
        filtered_probs = probabilities[valid_range]

        if len(filtered_sizes) > 1:  # Fit only if sufficient data
            # Linear regression in log-log space
            log_sizes = np.log10(filtered_sizes)
            log_probs = np.log10(filtered_probs)
            slope, intercept, _, _, _ = linregress(log_sizes, log_probs)

            # Calculate the fitted regression line in log-log space
            regression_line = slope * log_sizes + intercept

            # Convert back to the original scale for plotting
            fitted_probs = 10 ** regression_line

            # Plot regression line
            plt.plot(filtered_sizes, fitted_probs, label=f'L={L} (Exponent={-slope:.2f})', linestyle='--')

            # Print exponent for verification
            print(f"L={L}: Slope = {slope:.4f}, Exponent = {-slope:.4f}")

    # Configure log-log scale
    plt.xscale('log')
    plt.yscale('log')
    plt.xlabel('Avalanche Size $s$')
    plt.ylabel('Probability $P(s, L)$')
    plt.title('Avalanche Size Probability $P(s, L)$ in Log-Log Scale')
    plt.legend()
    plt.grid(True, which='both', ls='--')

    # Save the plot
    plt.savefig(f'probability_{name}.png')
    plt.show()

L = 6
T = 50

# Simulations for different system sizes
L_values = [64, 128, 256]
avalanche_data = [oslo_model(L, T) for L in L_values]

# Compute and plot probabilities
avalanche_probability(avalanche_data, L_values, 'one')
```

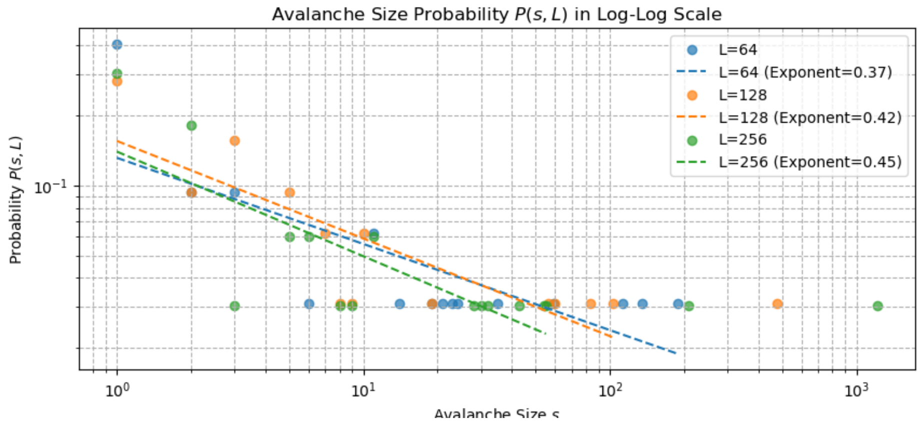

Figure 4: Plot for $L = 6$, $T = 50$, and $L \in \{64, 128, 256\}(0.4s)$

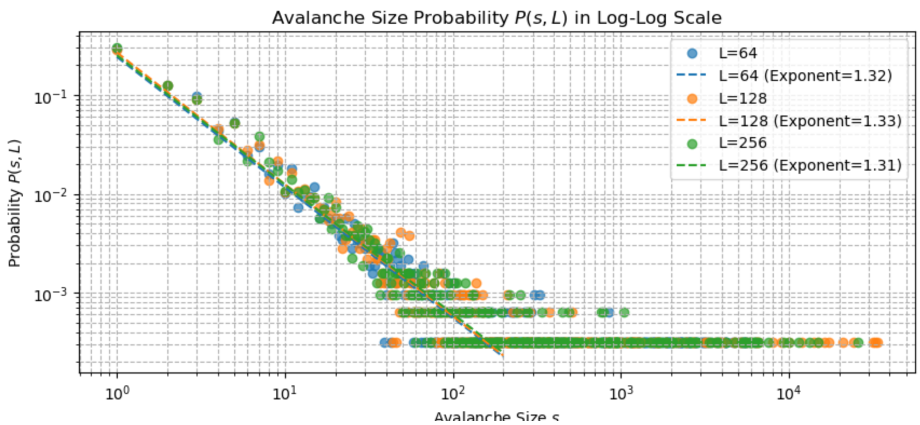

Figure 5: Plot for $L = 64$, $T = 5000$, and $L \in \{64, 128, 256\}(6.8s)$

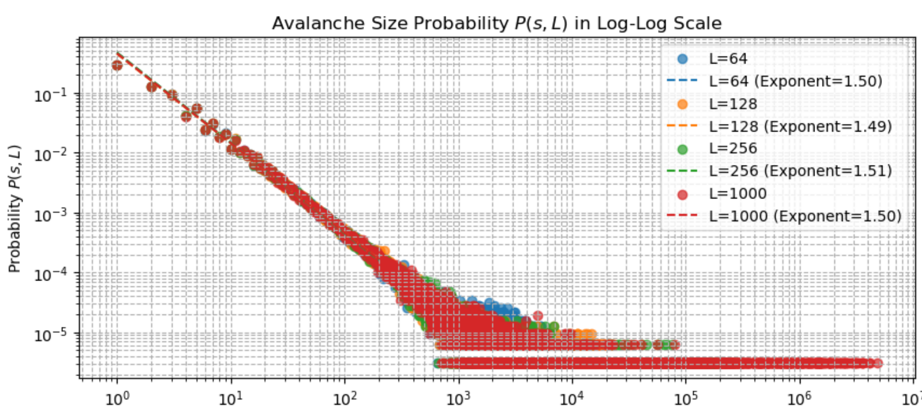

Figure 6: Plot for $L = 64$, $T = 500000$, and $L \in \{64, 128, 256, 1000\}(54m58.9s)$

Do you observe power law behavior in the avalanche size distribution?

ANSWER: Yes, I observe a power law behavior in the avalanche size distribution for the Oslo model, once the system has reached self-organized criticality (SOC).

Why does this power law break for large s?

ANSWER: The power law breaks down for large s because of finite system size (limited number of sites), saturation of the system, statistical sampling limitations, and time constraints. The probability of very large avalanches is reduced, leading to a cutoff in the distribution.