Complex Physics
Mid Term Exam2

Yu Liang Msc in Statistics crz342@alumni.ku.dk

Contents

1 avalanche-time

Core Implementation of the Model

0. **Model Initialization:**

```
    grid = np.zeros((L, L), dtype=int)
```

1. **Requirement:** The system is excited by adding a grain to some random position (x, y).

```
    def add_grain(grid, L):
    x, y = np.random.randint(0, L, size=2)
    grid[x, y] += 1
    return x, y
```

2. **Requirement:** Grains are only added after the entire system is relaxed, i.e., all $h_x, h < 2$.

```
    for step in range(max_steps):
    # Randomly add a grain
    add_grain(grid, L)
    # Handle collapse until the system stabilizes
    total_avalanche_size, max_extent = collapse(grid, L)


    def collapse(grid, L):
    while np.max(grid > 1):
    # Find all lattice points that need to collapse and handle relaxation
    # ...
```

This ensures that relaxation continues until no sites have $h > 1$.

3. **Requirement:** When the system is excited, it is relaxed in a sequence of relaxation moves where sites with $h_i > 1$ relax. The avalanche size is defined as the number of single-site relaxations until the new relaxed state is reached.

```
    def collapse(grid, L):
    total_avalanche_size = 0
    while np.max(grid > 1):
    topple_sites = np.argwhere(grid > 1)  # Find all lattice points that
    need to collapse
    total_avalanche_size += len(topple_sites)  # avalanche size definition
```

4. **Requirement:** This is done by moving two grains randomly to the four nearest neighbors of the excited site.

```
    # Define four directions (dx, dy)
    directions = np.array([[-1, 0], [1, 0], [0, -1], [0, 1]])  # up, down,
    left, right
    while np.max(grid > 1):
    for site in topple_sites:
    x, y = site
```

```
grid[x, y] -= 2  # Reduce two grains at the current lattice point
# Randomly distribute the two grains to four neighboring lattice points
for _ in range(2):
direction = np.random.randint(4)
dx, dy = directions[direction]  # Get direction change values
new_x, new_y = x + dx, y + dy
```

5. **Requirement:** Grains are moved out of the system when the excited site is on the boundary of the lattice.

```
# Check if the new position is within the boundary
if 0 <= new_x < L and 0 <= new_y < L:
grid[new_x, new_y] += 1
else:
# If outside the boundary, the grains overflow and are no longer part of
 the system
pass
```

6. **Suggestion:** Synchronous toppling, where all grains with $h > 1$ topple at the same time.

```
while np.max(grid > 1):
for site in topple_sites:  # Extract all topple sites at once, process
them, update grid, and then scan for all h > 1
```

The Sequence of Avalanches as a Function of Time

To study the sequence of avalanches over time, one should first observe the avalanches after many grains have been added, and the system has self-organized to reach the critical point.

For each step in the simulation:

```
for step in range(max_steps):
if step >= n_0:
# Only record the data after n_0 steps
```

Initially, n_0 is set to 0. As shown in Fig. 1, for smaller grid sizes like $L = 25$ and $L = 50$, the system reaches the self-organized state more quickly. However, for larger values of L, more grains need to be added before the system reaches the critical point. Because observing the distribution of avalanches at individual time steps does not provide clear insight, to convince oneself that the system has reached the critical point, a moving average method is used:

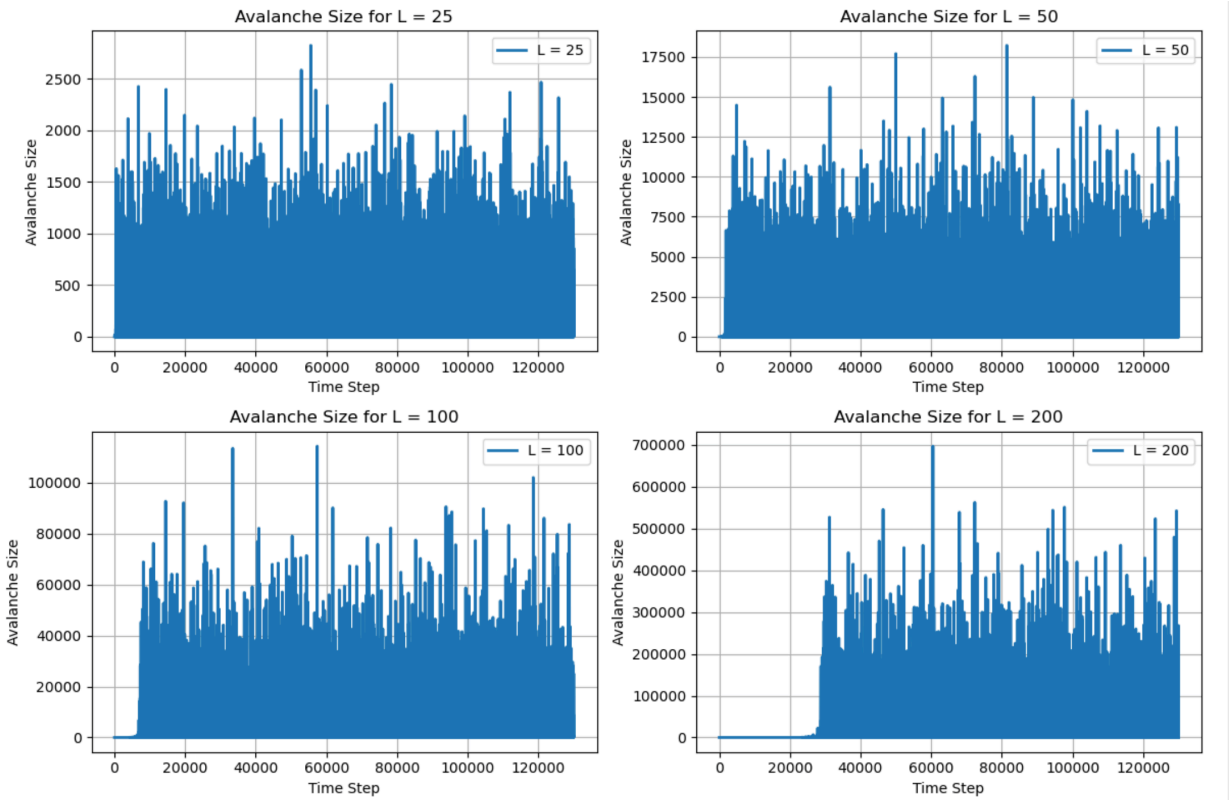

Figure 1: System evolution for different grid sizes $L = 25, 50, 100, 200$

Code for Moving Average Calculation:

```
# Compute moving average
def moving_average(data, window_size):
return np.convolve(data, np.ones(window_size) / window_size, mode='valid
')
```

In this implementation, the function 'np.convolve' is used to compute the moving average. Convolution takes two arrays and combines them by sliding one array (the window) over the other (the data) and producing a new array where each element is the weighted sum of the input data points inside the window. In this case, the window is uniform (all ones) and normalized by its size, so it computes the average within the window at each step.

As shown in Fig. 2, with a window size of 500, after approximately 30,000 steps, all systems reach a clear steady state.

For subsequent grid simulations in Q1/Q2, we will use $n_{\max} = 130,000$ and $n_0 = 30,000$ (discarding the first 30,000 steps). The analysis will focus on the results from the final 100,000 steps, where the system has reached a steady state. For Q3, with a grid size of 400, and Q4, where the addition method is changed, the values of $n_{\max}$ and n_0 are determined using the same method.

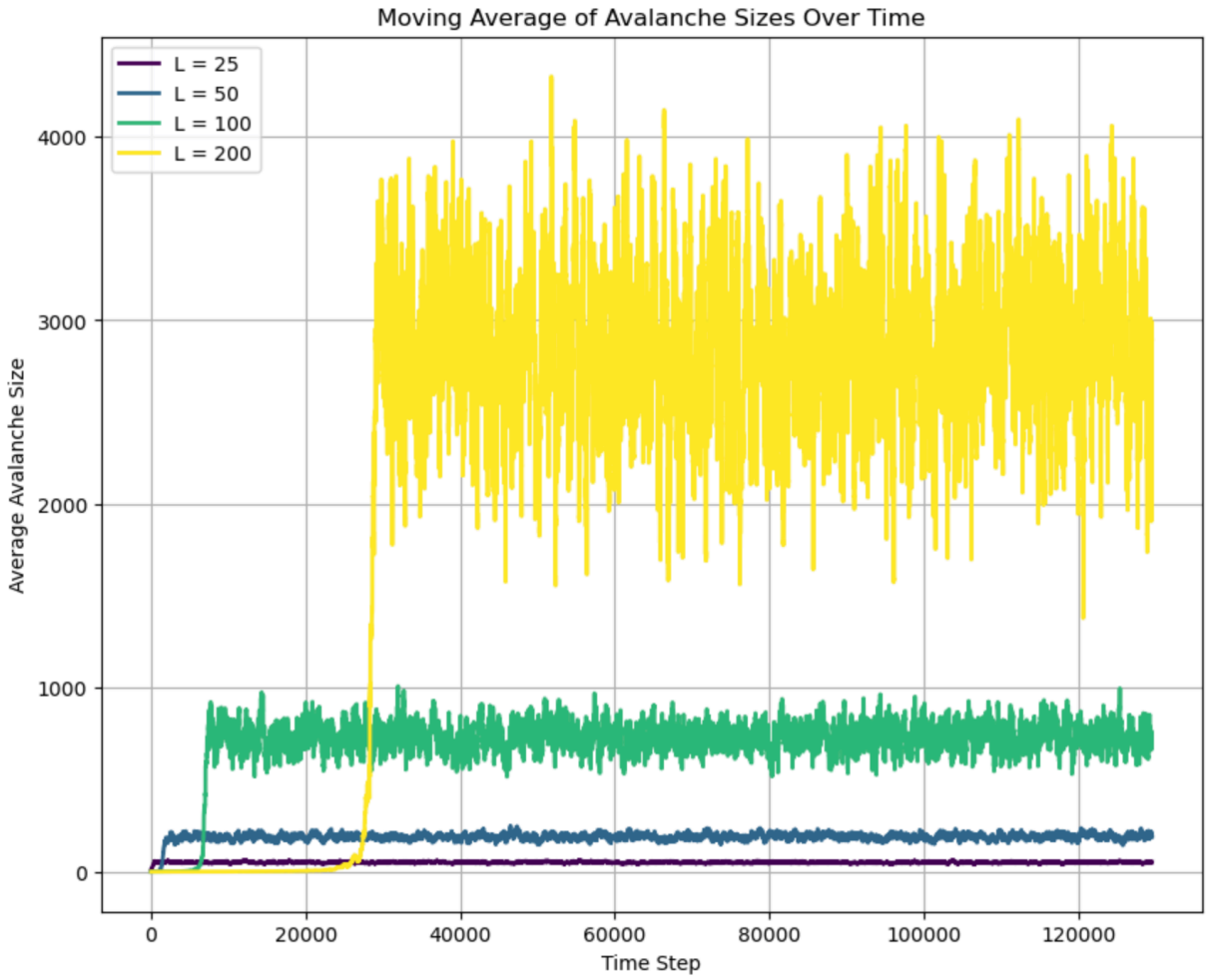

Figure 2: Moving average of avalanche sizes (window size = 500)

2 avalanche size distribution-different L

exponent for the size distribution

The probability distribution of avalanche sizes follows a power-law behavior with a cutoff for large sizes, given by:

$$P(s) = \frac{1}{s^{\tau}} \cdot f\left(\frac{s}{L^{D}}\right)$$

where $f\left(\frac{s}{L^{D}}\right) \sim 1$ for $s \ll L^{D}$, and f decays rapidly when the avalanche size approaches the system size $s > L^{D}$. Starting from the power-law relation $P(s) \sim s^{-\tau}$, taking the logarithm of both sides gives $\log P(s) \sim -\tau \log(s)$, showing a linear relationship with slope $-\tau$.

By plotting the avalanche size distribution on a log-log scale(Figure 3), we observe that as the avalanche size s increases, the frequency of occurrence decreases. For each system size L, there is a sharp cutoff beyond which the frequency drops dramatically. This cutoff corresponds to $s \sim L^D$. We also plot a power-law curve on the same graph and adjust the value of the exponent τ to match the curve with the data in the regime $s \ll L^D$. Through this visual estimation, we find that $\tau \approx 1.26$.

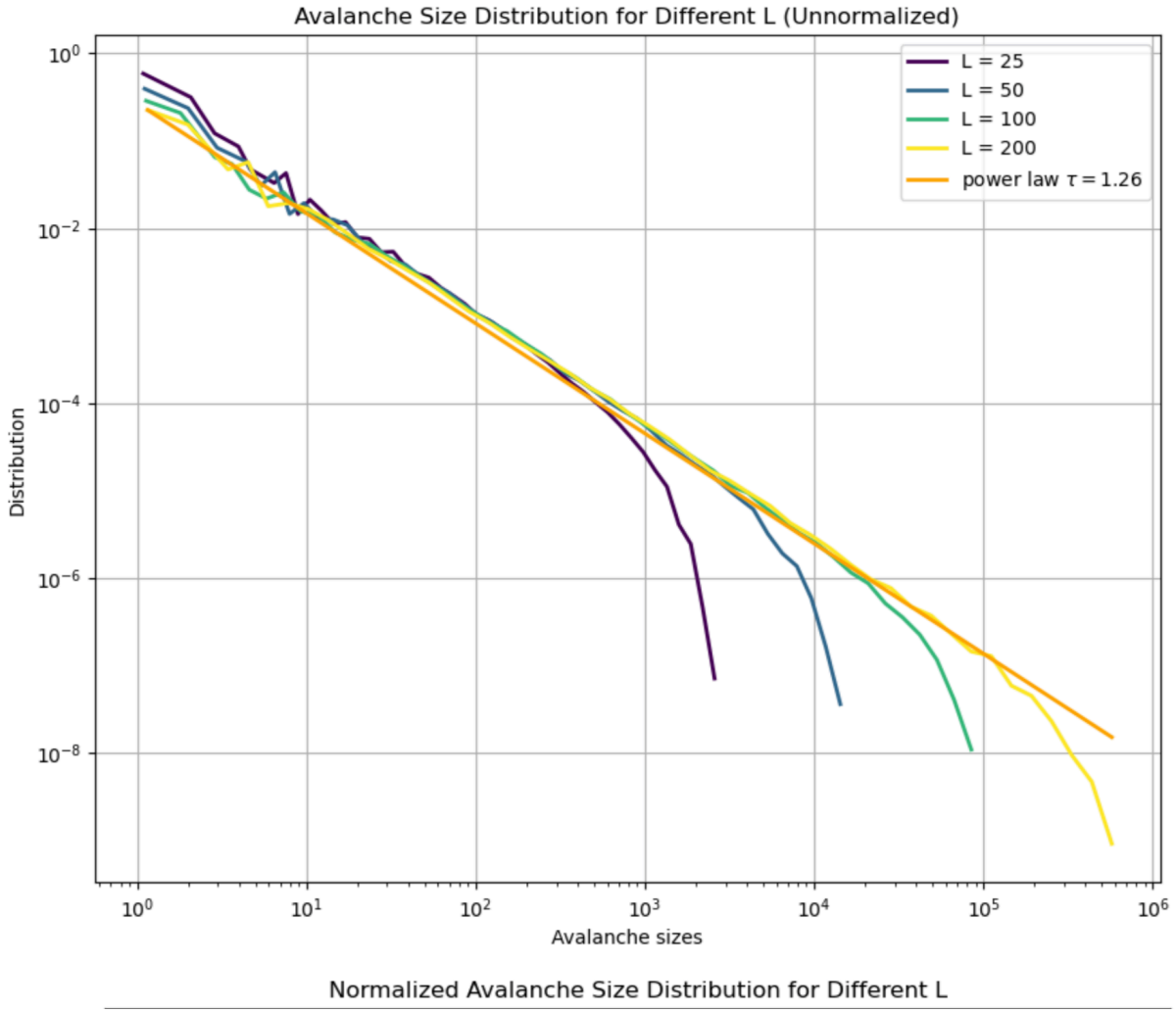

Figure 3: Avalanche size distribution for different system sizes L.

Next, we perform a more rigorous regression analysis to estimate the exponent.

```
# Define the size ranges for different L values
size_ranges = {25: (0, 700), 50: (0, 700), 100: (0, 700), 200: (0, 700)}

# Initialize empty list to hold all filtered sizes
all_filtered_sizes = []
```

```
    # Filter and aggregate data from all system sizes
    for L in L_values:
    avalanche_sizes = avalanche_sizes_dict[L]
    min_size, max_size = size_ranges.get(L, (1, np.inf))  # Default range if
    L is not in size_ranges

    # Filter out avalanche sizes of 0 and outside the specified range
    filtered_sizes = avalanche_sizes[(avalanche_sizes > 0) & (
avalanche_sizes >= min_size) & (avalanche_sizes <= max_size)]
    all_filtered_sizes.extend(filtered_sizes)

    # Count frequencies of all filtered sizes
    size_counts = Counter(all_filtered_sizes)
    total_avalanches = sum(size_counts.values())

    if total_avalanches == 0:
    return np.nan, np.nan  # If no avalanches in the range, return NaN

    # Create list of [size, frequency]
    size_freq_list = np.array([[size, count / total_avalanches] for size,
   count in size_counts.items()])

    # Take log10 of both size and frequency
    log_sizes = np.log10(size_freq_list[:, 0])
    log_freqs = np.log10(size_freq_list[:, 1])

    # Perform linear regression on log-log data
    slope, intercept, r_value, p_value, std_err = stats.linregress(log_sizes
   , log_freqs)
```

In this part of the analysis, we only consider the avalanche sizes in the range $(1, 700)$ (removing 0 since taking the log would result in negative infinity). We limit the range to 700 because, based on the previous graph, when size < 700, all L-values exhibit power-law behavior. When the size exceeds 700, the frequency of occurrences less than 10^{-4}. Including points with very low frequencies reduces the R^2 value of the regression, indicating a poor fit.

Thus, we aggregate the first 700 data points across the four L values and perform the regression analysis. The fitted slope is -1.26, and the R^2-value is 0.985.

exponent for the cutoff with L

The figure 4 below shows the normalized avalanche size distribution for different system sizes L. The x-axis represents the normalized avalanche size $s/L^{2.7}$, while the y-axis represents $P(s) \cdot s^{1.26}$. After determining the exponent $\tau = 1.26$, we observe that the first half of the curves for different L values are approximately flat. This corresponds to the theoretical expectation $P(s) \cdot s^{\tau} \sim 1$ when $s \ll L^D$.

To estimate the value of D, I started by testing values close to 2. As I incremented D, I found that at $D = 2.7$, the drop-off in the tail of the curves aligned perfectly for all system

sizes, as shown in Figure 4. Deviating from $D = 2.7$, either larger or smaller, causes the curves for different system sizes to diverge, with larger systems showing a wider and slower drop-off. At $D = 2.7$, the curves overlap, indicating that the system size dependency has been removed and the power-law scaling for avalanches is dimensionally correct.

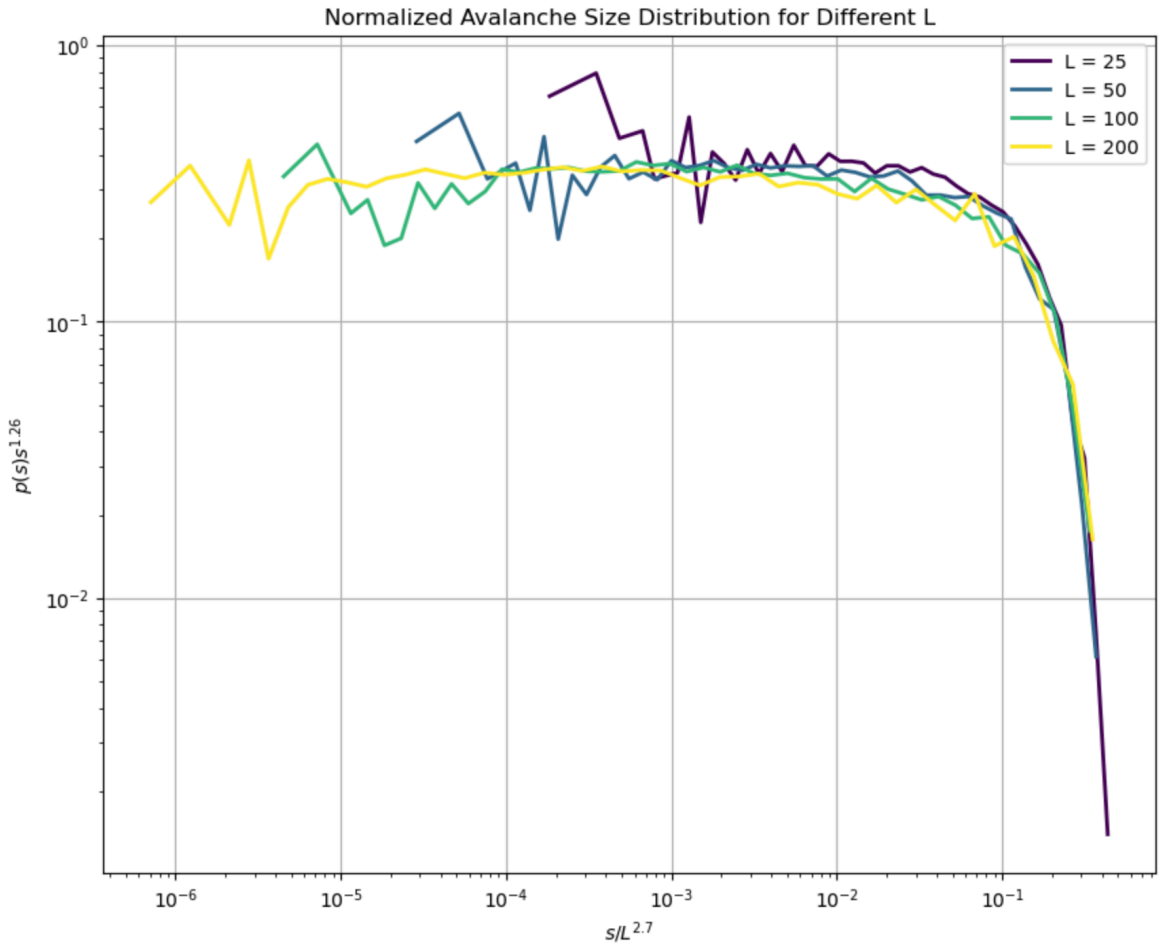

Figure 4: Collapse of the avalanche size distribution for different system sizes, showing the optimal scaling at $D = 2.7$.

3 Avalanche Sizes and Extension

Avalanche Sizes and Extension

In the program, the largest linear dimension of each avalanche (i.e., the longest extension along either the x-axis or y-axis) is calculated. The avalanche continues to collapse as long as there are lattice points with more than one grain.

```
    while np.max(grid > 1):
```

```
    topple_sites = np.argwhere(grid > 1)  # Find all lattice points that
   need to collapse
    total_avalanche_size += len(topple_sites)

    # Update the maximum extent length along the x and y axes
    x_min, x_max = topple_sites[:, 0].min(), topple_sites[:, 0].max()
    y_min, y_max = topple_sites[:, 1].min(), y_max = topple_sites[:, 1].max
   ()

    max_x_extent = max(max_x_extent, 1 + x_max - x_min)
    max_y_extent = max(max_y_extent, 1 + y_max - y_min)
```

If the avalanche is confined within a small portion of the grid, the extension is considered 1.

Regression Analysis

I selected a lattice size of $L = 400$, and the number of topplings in the avalanche scales with its horizontal extension, $s \sim L^D$. I applied the same method as before by taking the logarithm of both sides and performing regression analysis.

First, I plotted a rough graph, as shown in Figure 5. From this graph, we can see that the relationship between small extensions and avalanche size is not as clear as it is for larger avalanches. In fact, If the extension is 1, this means that no avalanche occurred; an extension of 2 indicates that two grains moved to the same side, but without causing a significant chain reaction. These small events do not produce the typical snowballing effect that significantly influences D (the avalanche dimension) and D_S (the fractal dimension).

Therefore, I set a lower bound(at least extension 30) for the extension values to exclude small extensions in regression.

Referring to the notes: *"Notice that D can be larger than two because each site can topple multiple times, corresponding to avalanches that get 'thicker' as they become horizontally larger."* Thus, setting too low of a bound will not capture this behavior, while setting too high of a bound would reduce the number of samples and lower the R^2 value of the regression due to the cutoff effect.

Finally, after performing the regression, the slope was found to be 2.7 with $R^2 = 0.93$.

Deduce the Dimension of Avalanche

Referring to the notes: *The steady-state condition implies that, on average, one avalanche must provide enough topplings to transport one grain to the boundary and out of the system. Since grains are added randomly, the average distance to the boundary is proportional to L, and because grains topple in random directions, the added grain has to participate in* $\sim L^2$ *steps before reaching the boundary.*

For $\tau < 2$, the steady state implies (inserting the cutoff function at the upper limit of the integral):

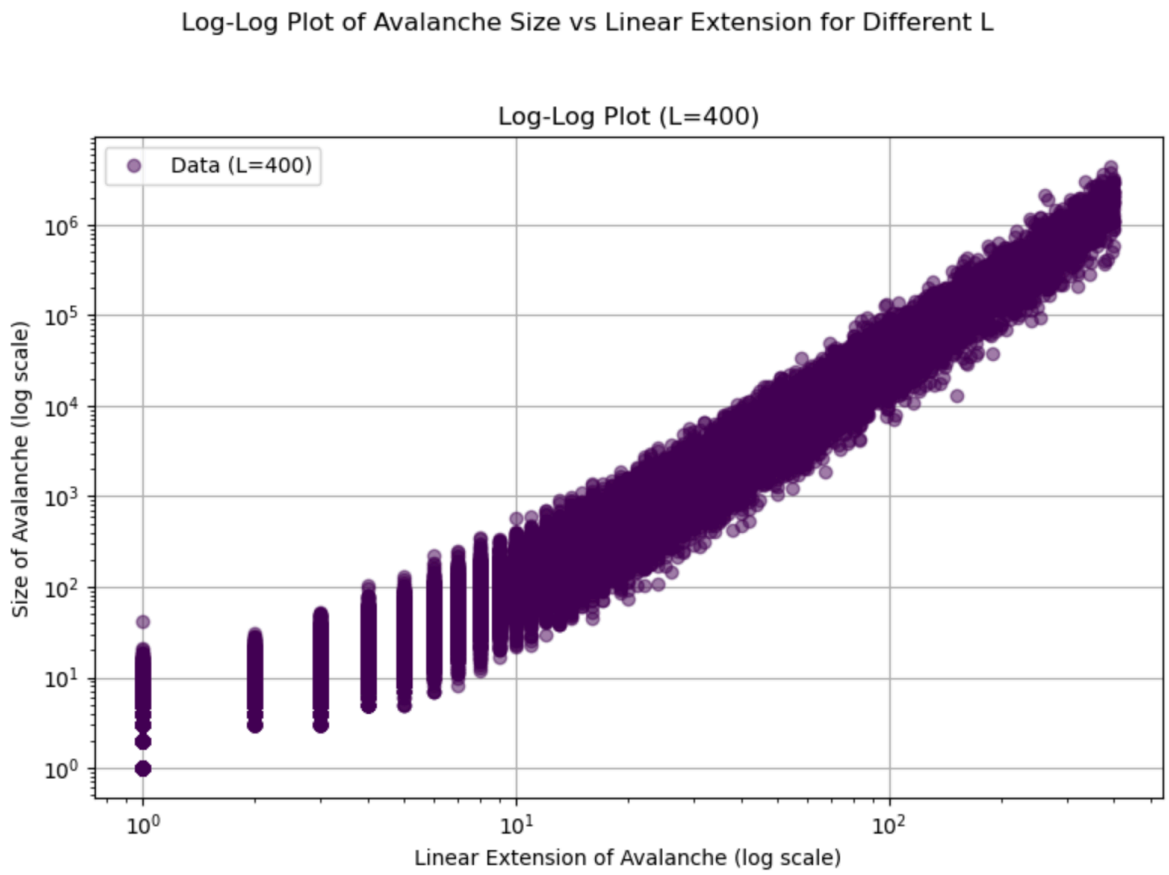

Figure 5: Log-log plot of avalanche size vs extension for $L = 400$, showing the scaling relationship between avalanche size and extension.

$$L^2 \sim \int_0^{\infty} s \cdot P(s)\, ds \sim \int_0^{L^D} s^{1-\tau}\, ds = L^{D\cdot(2-\tau)}$$

Thus, we obtain:

$$2 = D \cdot (2 - \tau) \Rightarrow \tau = 2 - \frac{2}{D}$$

The value of $D = 2.7$, obtained in both question 2 and 3, is consistent with $\tau = 1.26$ in question 2(and also in q3,see the attached codes,similar regression results)

The fractal dimension D_S can be estimated using the following relation:

$$(\tau - 1) \cdot D = 2 - D_S \Rightarrow \tau = 1 + \frac{2 - D_S}{D} \sim 1.26, \quad D = 2.7 \quad \Rightarrow D_S \approx 1.3$$

This calculation shows that the fractal dimension D_S is approximately 1.3.

4 Avalanche Sizes: Edge Addition

Edge Addition Code

The following Python function adds a grain randomly on the edge of the grid:

```
max_steps = 3000000  # Maximum number of steps,this time have to be
larger
n_0 = 2000000  # Start recording avalanche events after step n_0
def add_grain_on_edge(grid, L):
edge_choice = np.random.randint(4)  # Randomly choose an edge, 0: top,
1: bottom, 2: left, 3: right

if edge_choice == 0:  # Top edge y = 0
x = np.random.randint(0, L)
y = 0
elif edge_choice == 1:  # Bottom edge y = L-1
x = np.random.randint(0, L)
y = L - 1
elif edge_choice == 2:  # Left edge x = 0
x = 0
y = np.random.randint(0, L)
else:  # Right edge x = L-1
x = L - 1
y = np.random.randint(0, L)

grid[x, y] += 1
return x, y
```

Exponent Estimation

We first visually estimated the values of $\tau = 1.66$ and $D = 2.75$ from the plots. As shown in Figures 6 and 7, the scaling behavior is evident when the curves are plotted. The slope in the unnormalized plot (Figure 6) corresponds to $\tau = 1.66$, and the normalized plot (Figure 7) shows the collapse of the curves for different system sizes at $D = 2.75$.(basically consistent with previous D=2.7)

To confirm these results, we applied regression analysis(similar analysis as in Q2), which returned the following values: Slope: -1.66 R^2: 0.99

5 boundary-driven avalanche

When grains are added only at the boundary, the probability of a grain exiting the system increases, reducing the likelihood of large avalanches. This leads to a steeper power-law exponent τ, as compared to random grain addition throughout the system, as shown in question 2. The narrower power-law curve for boundary addition indicates fewer large avalanches.

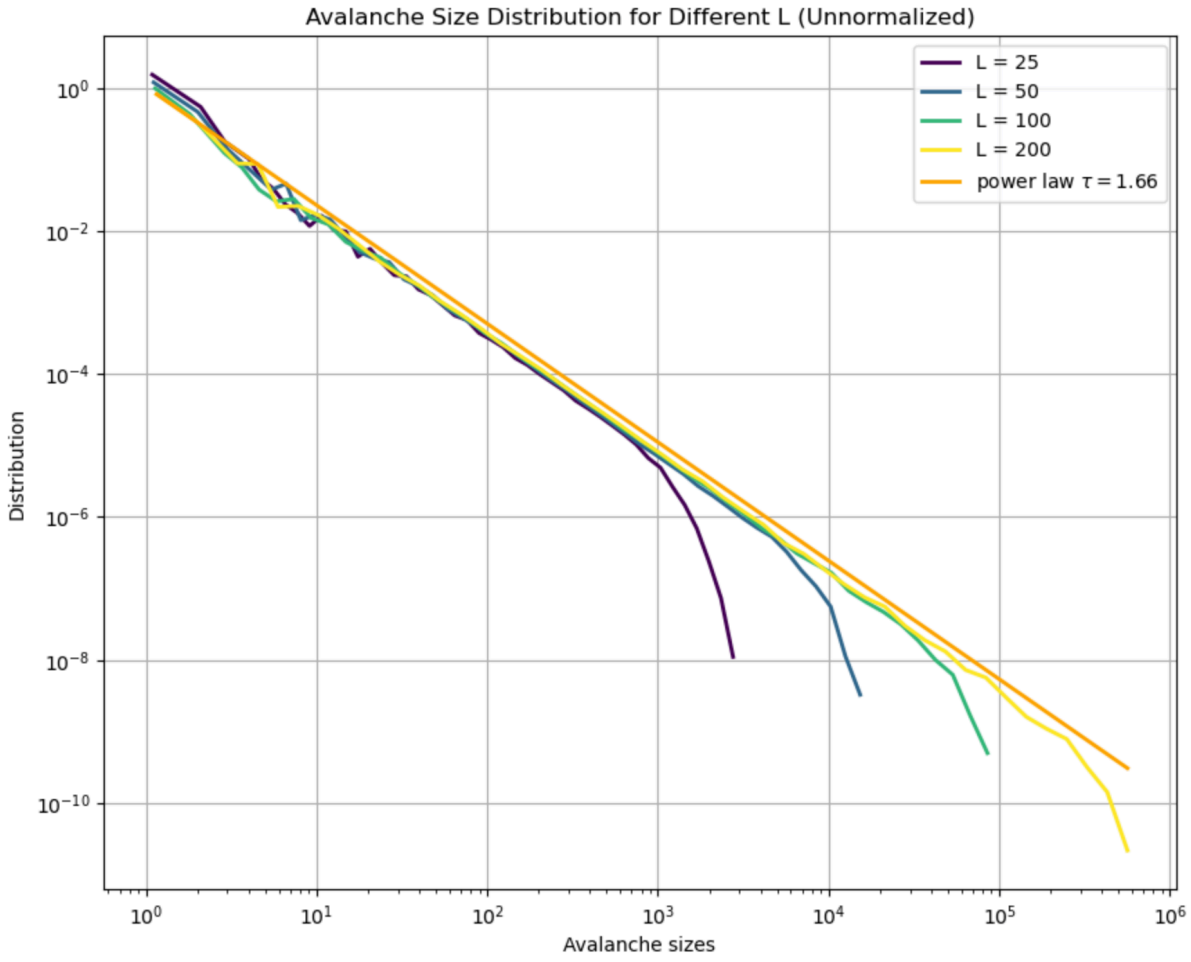

Figure 6: Unnormalized avalanche size distribution showing the power-law behavior with $\tau = 1.66$.

From the lecture notes, we understand that when grains are added randomly at the boundary, on average it only takes L steps for a grain to exit the system, rather than L^2 steps, because the exit is much closer. This is derived from the first return time to the boundary of added grains:

$$\text{Average exit time from boundary} = \int_0^{L^2} \frac{t \cdot dt}{t^{3/2}} \propto L$$

$$L \sim \int_0^{\infty} s \cdot P(s)\, ds \sim \int_0^{L^D} s^{1-\tau}\, ds = L^{D\cdot(2-\tau)}$$

From this, we obtain the relation:

$$1 = D \cdot (2 - \tau) \quad \Rightarrow \quad \tau = 2 - \frac{1}{D}$$

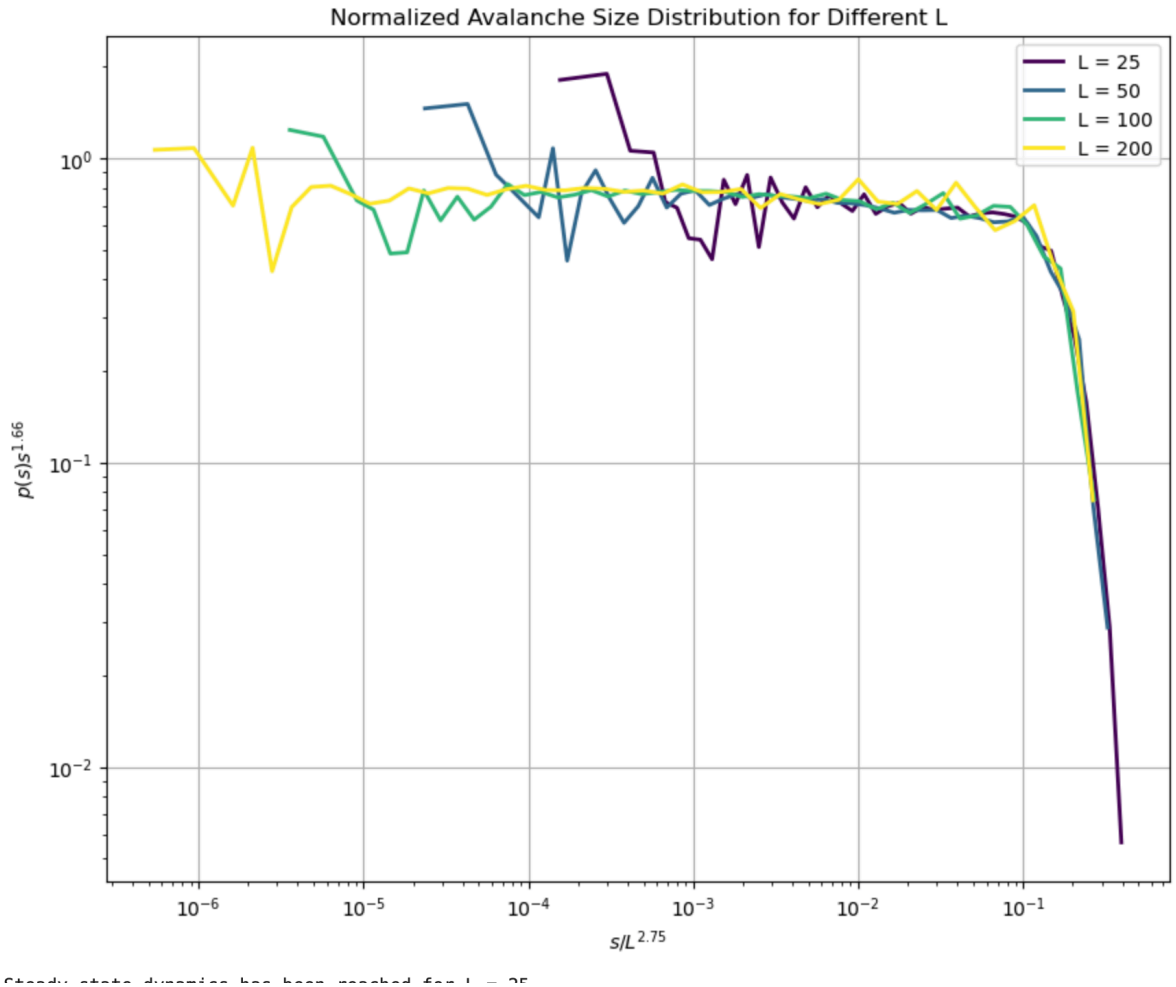

Figure 7: Normalized avalanche size distribution, showing the scaling with $D = 2.75$.

Substituting $D = 2.75$ into this equation:

$$\tau = 2 - \frac{1}{2.75} \approx 1.64$$

This value is consistent with the previously estimated $\tau = 1.66$, confirming that our scaling arguments and regression analysis results are in agreement. Therefore, both $D = 2.75$ and $\tau = 1.66$ are validated by the theoretical framework.

Appendix

References:

Kim Christensen, Nicholas R. Moloney. *Complexity and Criticality* (English reprint). Kim Sneppen. *Complex Physics*