Mini - Project: Forest Fire Model

Sanjana Gupta
PHY-3150: Computational Physics
Professor: Somendra Mohan Bhattacharjee
Submitted: November 28, 2023

1 Forest Fire Model and Self Organised Criticality

Self-organized criticality (SOC) characterizes dynamic systems converging toward a critical point, functioning as an attractor. This state entails the system's inclination to evolve into a condition where minor alterations can trigger widespread effects, irrespective of the system's size or temporal scale. Such effects often align with power laws or fractals, mathematical constructs recurring across varying scales. SOC stands as a fundamental mechanism driving complexity in natural systems.

The essence of self-organized criticality lies in systems reaching a critical state devoid of external calibration or control. This critical juncture signifies a heightened sensitivity to minute changes, leading to large-scale ramifications following power laws. Power laws, mathematical models, delineate event frequency in relation to their magnitude.

The forest fire model embodies a rudimentary system capable of attaining a critical state via straightforward directives:

- Random tree growth within empty spaces with a given probability.
- Occasional ignition of trees by random sparks with a specified probability.
- Fire propagation to neighboring trees, leading to their combustion.

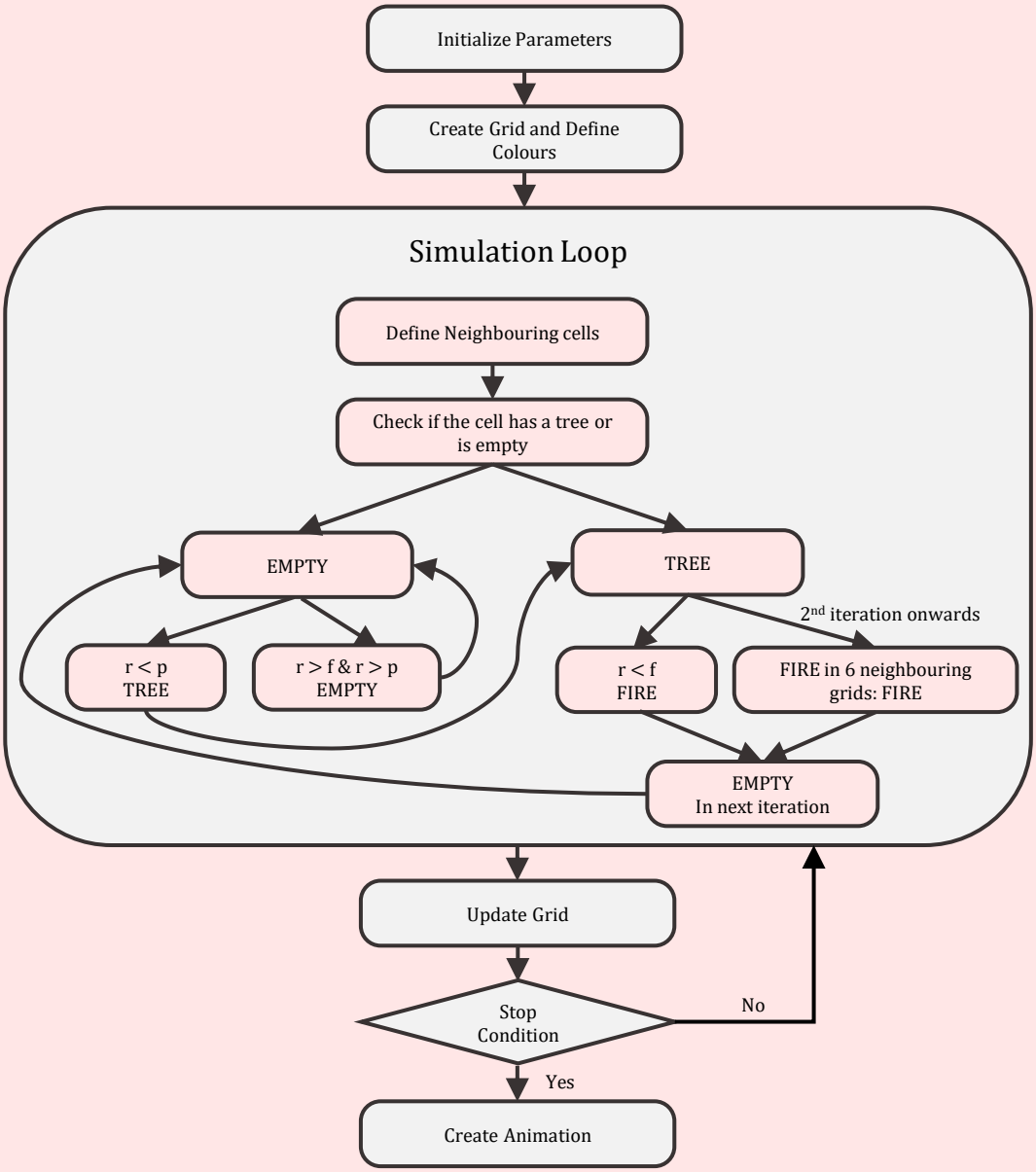
These rules establish a cyclic interplay between forest growth and its destruction. Increased tree density augments the likelihood of fire ignition and expansion. Inversely, intensified fire leads to increased tree incineration and forest clearance. The system can achieve equilibrium between these processes, representing the critical state.

This critical phase is of interest as it generates fires of diverse sizes adhering to a power law distribution. Such distributions entail numerous small-scale fires and fewer larger ones, with fire frequency decreasing as size amplifies. The power law's universality across system sizes and temporal scales denotes its presence within a small or extensive grid, over brief or protracted durations. Additionally, the power law remains invariant to system parameters, like tree growth and fire ignition probabilities, within specified limits.

Moreover, the system's critical phase yields fires displaying fractal geometry. Fractals, characterized by self-similarity, depict object forms resembling themselves across various scales. For instance, snowflakes exhibit fractal geometry by showcasing similar branching patterns across magnification

levels. Fires in the forest fire model embody fractals as they exhibit cluster shapes recurring across different scales. This fractal geometry mirrors the power law governing fire sizes, signifying the system's scale invariance.

Algorithm and Code:
The major change in a triangular or hexagonal lattice compared to a square lattice is that the number of neighbours becomes 6. I have written a code that creates a simulation of a forest fire using a grid-based model similar to the cellular automaton model in Python. It initializes a grid representing a forest area with cells classified as empty land, trees, or cells currently on fire. The simulation operates based on predefined probabilities for tree growth and fire ignition. The `update` function manages the rules governing the spread of fire across the grid: if a cell is on fire, it can ignite neighboring trees, and trees can catch fire based on proximity to existing fires or a specified probability. The animation is generated using Matplotlib's `FuncAnimation`, which continuously updates the plot according to the fire simulation. The resulting animation illustrates the evolving dynamics of the fire's propagation across the forest landscape. Additionally, the code saves the generated animation as a GIF file for later viewing or analysis.
We can use the following flowchart for the algorithm:
Let the random probability be r, the probability of tree growth bt p, and the probability of fire be f:



The algorithm can be implemented in Python as follows:

Listing 1: Importing the required libraries

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import animation
from matplotlib import colors
from matplotlib import colors as mcolors
from matplotlib.animation import FuncAnimation
from matplotlib.colors import ListedColormap
from tqdm import tqdm
from scipy.optimize import curve_fit
plt.rcParams['figure.dpi'] = 200
plt.rcParams['font.family'] = 'serif'
```

```
plt.rcParams['font.serif'] = ['Times New Roman'] + plt.rcParams['font.serif']
```

Listing 2: Forest Fire Animation

```
%matplotlib notebook

EMPTY = 0
TREE = 1
FIRE = 2

rows = 100
cols = 100
prob_tree_growth = 0.05                                    # Probability of tree growth for an empty cell
prob_fire = 0.01                                           # Probability of a fire starting in a tree cell

tree_probability = np.random.uniform(size=(rows, cols))
grid = np.where(tree_probability < 0.5, EMPTY, TREE)

colors_list = ['saddlebrown', 'yellowgreen', 'orange']
cmap = ListedColormap(colors_list)

fig, ax = plt.subplots()
mat = ax.matshow(grid, cmap=cmap, vmin=0, vmax=2)
ax.axis('off')

def update(frame_number):
    global grid
    new_grid = grid.copy()
    for i in range(rows):
        for j in range(cols):
            if grid[i, j] == FIRE:
                new_grid[i, j] = EMPTY
            elif grid[i, j] == TREE:
                neighbors = [
                    grid[(i - 1) % rows, j],               # Upper neighbor
                    grid[(i + 1) % rows, j],               # Lower neighbor
                    grid[i, (j - 1) % cols],               # Left neighbor
                    grid[i, (j + 1) % cols],               # Right neighbor
                    grid[(i - 1) % rows, (j + 1) % cols],  # Top-right neighbor
                    grid[(i + 1) % rows, (j - 1) % cols]   # Bottom-left neighbor
```

```
                ]
                if FIRE in neighbors or np.random.uniform() < prob_fire:
                    new_grid[i, j] = FIRE
            elif grid[i, j] == EMPTY and np.random.uniform() <
                prob_tree_growth:
                new_grid[i, j] = TREE
    grid = new_grid
    mat.set_data(grid)
    return [mat]

ani = animation.FuncAnimation(fig, update, interval=50)
ani.save(
    r"D:\Semester 5\Computational Physics\Assignments\forest_fire.gif",
    writer='pillow', dpi=300, fps=10
)
plt.show()
```

We also do some further analysis by calculating the area covered by fire for different values of probability of tree growth.

Listing 3: Further Analysis

```
def simulate_fire(rows, cols, prob_tree_growth, prob_fire, iterations):
    tree_probability = np.random.uniform(size=(rows, cols))
    grid = np.where(tree_probability < (1 - prob_tree_growth), EMPTY, TREE)

    for _ in range(iterations):
        new_grid = grid.copy()
        for i in range(rows):
            for j in range(cols):
                if grid[i, j] == FIRE:
                    new_grid[i, j] = EMPTY
                elif grid[i, j] == TREE:
                    neighbors = [
                        grid[(i - 1) % rows, j],                # Upper
                            neighbor
                        grid[(i + 1) % rows, j],                # Lower
                            neighbor
                        grid[i, (j - 1) % cols],                # Left
                            neighbor
                        grid[i, (j + 1) % cols],                # Right
                            neighbor
                        grid[(i - 1) % rows, (j + 1) % cols],   # Top-right
                            neighbor
                        grid[(i + 1) % rows, (j - 1) % cols]    # Bottom-
                            left neighbor
                    ]
                    if FIRE in neighbors or np.random.uniform() < prob_fire
                        :
```

```
                        new_grid[i, j] = FIRE
        grid = new_grid

    return grid

def measure_fire_extent(grid):
    return np.count_nonzero(grid == FIRE)

# Parameters
prob_fire = 0.01
iterations = 10

# Varying probabilities of tree growth
prob_tree_values = np.linspace(0.01, 0.1, 10)
fire_extents = []

for prob_tree_growth in tqdm(prob_tree_values, desc='Simulating Fires'):
    final_state = simulate_fire(rows, cols, prob_tree_growth, prob_fire,
        iterations)
    fire_extent = measure_fire_extent(final_state)
    fire_extents.append(fire_extent)

# Plotting
plt.figure(figsize=(8, 6))
plt.plot(prob_tree_values, fire_extents, 'ko', markersize=3, label='
    Simulated Data')
plt.xlabel('Tree Growth Probability')
plt.ylabel('Fire Extent [grids]')
plt.grid(True)

# Defining the power-law function
def power_law(x, a, b):
    return a * np.power(x, b)

# Fitting the power-law function to the data
fit_params, _ = curve_fit(power_law, prob_tree_values, fire_extents)
plt.plot(prob_tree_values, power_law(prob_tree_values, *fit_params), 'r-',
    label='Power Law Fit')
plt.legend()
plt.savefig(f"D:\Semester 5\Computational Physics\Assignments\power_law.pdf
    ", dpi=300,
                    bbox_inches="tight", transparent=True)
plt.show()
```

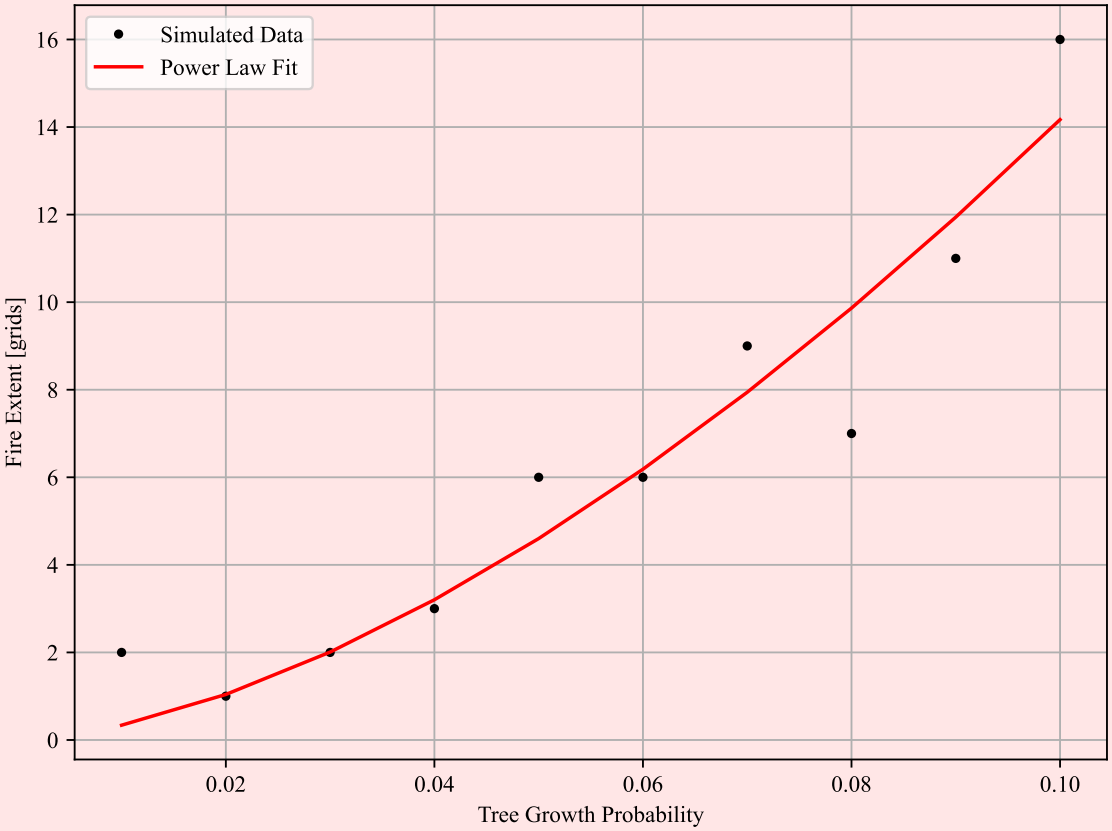


Figure 1: Fire extent vs. Tree growth Probability shows an increasing trend as expected. The higher the tree growth probability, the more likely the forest is dense with trees, which makes it easier for the fire to spread.

2 References

contributors, Wikipedia. *Forest-fire model*, 2023. https://en.wikipedia.org/wiki/Forest-fire_model.

Hill, Christian. *The forest fire model*, 2016. https://scipython.com/blog/the-forest-fire-model/.

Lee, Sang-Hoon, Seong-Yun Kim, Seung-Yun Lee, and Jong-Hyun Park. "Forest fire model with a new neighborhood structure." *Frontiers in Physics* 8 (2020): 257. https://www.frontiersin.org/articles/10.3389/fphy.2020.00257/full.

Class Notes - Computational Physics - Monsoon 2023, Ashoka University

Python Code used for this Mini-Project can be found here: Python Code