The University of Melbourne
COMP90072 - The Art of Scientific Computing

COMPLEX SYSTEMS

Adam Monteleone
Student ID: 913516

June 10, 2022

Abstract

Using a recursive algorithm we created a simulation in python 3.9 of the original Bak-Tang-Weisenfeld model as outlined in [2] and showed that the simulation of the model indeed exhibits self organized criticality. Additionally we use the simulation to monitor avalanche observables such as area, topples, length and loss. Each observable (with the exception of loss) was found to follow a power law in the critical state confirming our theoretical predictions..

The abelian sandpile model was then extended to incorporate more advanced toppling methods such as uniform toppling and gradient toppling where we consider the trade off between realism and computational time complexity. The structure of the grid was then varied between the cylinder, Möbius band, torus and Klein bottle. Where we observe large changes in the extreme value statistics for avalanche events depending on the grid topology. The limitations of determining avalanche distances in curved spaces is also discussed.

Finally the extended sandpile model was modified to enable earthquake modelling. The earthquake model was then shown to also exhibit self organized criticality with the avalanche observables (excluding loss) taking the form of power laws. The data for all earthquakes in the US from 2019-2022 is scraped from the internet and shown to empirically follow the Gutenberg-Richter law having a b value of exactly 1.00. Using our earthquake model we then varied our parameter α to coincide with the empirical distribution of the scraped data and hence deduced that the data implies a stress energy dissipation rate of 7.75%.

Contents

Chapter 1

Introduction

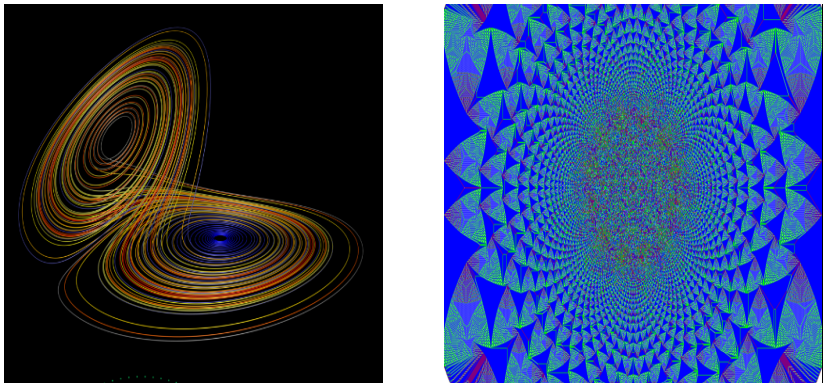
In the natural world we frequently encounter systems that possess behaviour that is too complicated to be described analytically. This may be because the state of the system depends on a large number of degrees of freedom or perhaps because the components of the system interact with each other in a non-linear way. That is to say that changes in cause and effect are not necessarily proportional. Systems with such characteristics are said to be 'complex'.

The study of complex systems is a young field of research which finds its roots in chaos theory. Chaotic systems were first seriously studied by Henri Poincaré in the late 19th century and are systems which have behaviour that is seemingly unpredictable. Chaotic systems are typically non-linear, deterministic in principle and have results that are highly sensitive to the choice of initial conditions. The most famous example of a chaotic system is the Lorenz system named after mathematician Edward Lorenz. The Lorenz system has the feature that it exhibits something called a strange attractor (for the Lorenz system this is called the Lorenz attractor: see figure 1.1). The Lorenz system is important for many reasons but one philosophical implication that should be emphasized is that you should not always assume that behaviour that appears chaotic is non-determined.

Chaotic systems can be viewed as a subset of the much larger class of complex systems. Since complex systems do not necessarily have to be chaotic or determinstic they can include system that are truly non-deterministic. Despite the vastness of phenomena the field encapsulates, significant progress has been made on theories of complex systems. Specifically in 1987 Per Bak, Chao Tang and Kurt Wiesenfeld together came up with the theory of *self organized criticality* as a way of explaining explaining 1/f noise. The paper [2] ended up being one of the most cited works in the theoretical physics literature, and was an attempt to define a phenomena observed in many slowly driven complex systems. Within this paper Bak et al. also demonstrated that a model with very simple deterministic rules can consistently exhibit this 'self organized criticality' phenomenon. The specific model they considered was known as the Baker-Tang-Weisenfeld Model or abelian sandpile model (since the model can be explained in terms of sand piles with abelian toppling). The model was captivating since it had very simple determinstic rules yet phenomena within it consistently exhibited complex behaviour and followed power laws after a certain 'critical state' had been reached.

Power laws are ubiquitous in nature and the abelian sandpile model as well as self organized criticality found many application throughout different disciplines. In particular one of the first applications this model saw was in helping model natural disasters such as landslides, avalanches, forest fires and earthquakes. Per Bak helped pioneer this research through publications such as [1] which explored the use of computer simulations to model Earthquake phenomena. Since then computers have become significantly more powerful as well as more cheap and as a result this type of modelling is now common place for geologists who specialise in natural disasters.

Inspired by the above, our aim is to first build the sandpile model ourselves to investigate such phenomena. Then if all goes well, we would like to adapt this model into an earthquake modelling simulation. However before we consider the implementation of such models a discussion of the necessary background theory is required.



(a) Lorentz Attractor (b) Scale invariance in the BTW Model

Figure 1.1

1.1 Self-Organized Criticality, Power Laws and Scale invariance

Self Organized Criticality (or SOC) was first introduced by Baker-Tang-Wisenfeld in 1987 as an explanation for the 1/f noise observed. Since then it has been more rigorously studied however currently there is no consistent definition of what SOC is in the literature. The definition of SOC we adopt here is sometimes known as weak SOC[1]
.

A system is said to exhibit **self organized criticality** if its phase space contains a strange attractor where events of all sizes can occur and where the size of the distribution of these events follows a power law. The non technical definition of an **attractor** is that it is a set of states in phase space which the systems evolves towards for almost all initial conditions. An attractor is said to be a **strange attractor** if it has a fractal structure/geometry.

Self organized criticality frequently occurs in systems that are being slowly driven. Such systems tend towards the critical state autonomously without the necessary fine tuning of parameters (hence the 'self' in self organized criticality).

Since a system that exhibits SOC has an attractor that is fractal there is no characteristic length scale associated to it and therefore we subsequently anticipate that such systems feature a kind of 'scale invariance'. **Scale invariance** is the property of a system to look the same no matter how much you zoom in or out. As we can see in 1.1b the abelian sandpile model exhibits scale invariance in the critical state. Scale invariance tends to manifest itself through the observables of the system following power laws.

Definition 1.1.1. *A random variable s is said to follow a **power law** if its probability density function $P(s)$ has the form*[2]

$$P(s) \sim s^{-b} \tag{1.1}$$

.

For the derivation of the fact that the scale invariance of an observable implies the function has the form of a power law see A.3.1 [3]

Although natural disasters such as earthquakes, landslides, and forest fires have been said to exhibit SOC [5] and thus exhibit observables governed by power laws a clarification needs to be made about the validity of such claims. In nature there cannot be true scale invariance, since at some point we will reach the atomic and then subatomic scale. So when such phenomena are described to be power laws really this is only true up to some cut off value. This is also reflected mathematically in the fact that area under the probability density function being to being normalized fixes a scale. Therefore in some sense it is probably more accurate to replace the power-law distribution with a Pareto Distribution that has the form of a power law up to some cut off value s_{min}. For a more thorough discussion see chapter 1 of [4]

$$P(s) = \begin{cases} (\frac{s}{s_{min}})^{-b}, & \text{if } s > s_{min} \\ 1, & \text{else} \end{cases} \tag{1.2}$$

[1] In the original definition of SOC there was an added requirement that the temporal signal of the system is pink noise or at least tends to pink noise in the limit of the system becoming infinite in size. (see[4] chapter 5)

[2] The $\sim$ here is used instead of '=' since the distribution is defined up to a normalization constant.

[3] No space to do the derivation :(

1.2 The Bak-Tang-Wiesenfeld/Abelian Sandpile Model

The ensuing explanation of the BTW model will not be overly mathematically formal for pedagogical reasons. Instead we explain the BTW model using idealized piles of sand on a lattice.

Consider a $N \times M$ dimensional lattice such that $p_t(i,j)$ is the number of sand grains located at site (i,j) at time t where $i \in \{1,..N\}$ and $j \in \{1,..M\}$ and $t \in \mathbb{N}_0$. At each time step t a new grain is added to a random site on the lattice such that over sand piles up over time.

If an individual pile of sand reaches a height of 4 or more, then the pile topples and distributes four grains of sand to the immediate 4 adjacent sites, mathematically this toppling algorithm can be stated as:

$$p_t(i,j) = p_t(i,j) - 4 \tag{1.3}$$
$$p_t(i \pm 1, j) = p_t(i \pm 1, j) + 1 \tag{1.4}$$
$$p_t(i, j \pm 1) = p_t(i, j \pm 1) + 1 \tag{1.5}$$

Additionally if for instance a toppling occurs on a corner then 2 of the 4 sand grains are said to be lost from the system. If a site is on an edge but not a corner only 1 of the 4 sand grains would be lost during a toppling.

The abelian sandpile exhibits self organized criticality as it is a slowly driven system. Where the driving is coming from the random drop of a single sand grain every time increment. Moreover since no site can have a value of 4 (or more generally a value equal to the threshold h) the total number of sand grains must steadily increase until the grid nears capacity. When its near its maximum capacity however the avalanche events will be very large and generate a lot of loss decreasing the total mass. This implies that the total mass as a function of time will oscillate about some point and that these states within the oscillation would correspond to the sand piles strange attractor. Thus we expect the model to a have a statistically stationary state with the mean height or mass density being the corresponding observable.

Since the lattice is finite we expect that the observables will only have power law behaviour up to some cutoff. This cutoff will be proportionate to the length of the grid so perhaps taking the square root of the grid area would be reasonable for the truncation of the range in which we expect the power laws to occur. However if we had an infinite grid such a truncation would not be necessary as we would have true scale invariance. Also note here that for an infinite grid loss will be zero so this suggests that loss is not scale invariant and therefore an observable which we do not expect to follow a power law distribution.

Considering the possibility of an infinite grid is clearly not computationally feasible since the computer only has finite memory and processing power. Not to mention the probability of dropping sand at a single point on an infinite grid will be of measure 0. If instead however we dropped sand on a finite subset of the infinite lattice then this latter issue is avoided. To make it computationally feasible though we could consider any part of the lattice not within that subset to be "out of bounds" and so we would not need to worry about storing all such sites. But wait! this is isomorphic to the abelian sandpile we were already going to implement. It is just we have no reinterpreted the lost sand grains as sand grains that are not off the board but in a region of the lattice which we consider to be out of bounds.

Now we can define the average height (equivalently the mass density ρ) of the sandpile to be given by

$$\langle p_t \rangle = \frac{1}{NM} \sum_{i=1}^{N} \sum_{j=1}^{M} p_t(i,j) \tag{1.6}$$

Given this we can try and estimate the mass density of the statistically stationary state that the total mass will oscillate about. After the system has reached a steady state it is reasonable to assume that a disproportionate number of the piles will have 1 less than the toppling height. A crude estimate of the upper bound might be supposing such a distribution follows a kind of sharp 80-20 rule. Thus we might have an upper bound given by $1\%, 4\%, 16\%, 79\%$ of the grid having heights 0,1,2 and 3 respectively. The lower bound might be uniformly distributed where the probability weights are 25%, 25%, 25%, 25%. Thus for a 10 x 10 grid if we took the arithmetic mean of our bounds we would then have the following expectation values:

$$\langle p_{\text{Upper}} \rangle = 0\left(\frac{1}{100}\right) + 1\left(\frac{4}{100}\right) + 2\left(\frac{16}{100}\right) + 3\left(\frac{79}{100}\right) = 2.74 \tag{1.7}$$
$$\langle p_{\text{Lower}} \rangle = 0\left(\frac{25}{100}\right) + 1\left(\frac{25}{100}\right) + 2\left(\frac{25}{100}\right) + 3\left(\frac{25}{100}\right) = 1.50 \tag{1.8}$$

The Arithmetic mean of such bounds yields an estimate of $\langle p_t \rangle = \frac{1.50+2.74}{2} = 2.12$.

Chapter 2

Simulating The Abelian Sandpile Model

2.1 Methodology

2.1.1 Setup

It was decided to implement the basic abelian sandpile Model using python 3.9 and making use of common data science and numerical computing libraries such as NumPy, Seaborn and Pandas. This has the benefit of containing a lot of methods and data structures that are well optimized since certain functions especially in NumPy are written in the lower level programming language C.

Viewing the sandpile as an object in its own right where its observables are attributes lends itself to naturally being a class object in python. For the most basic sandpile the user needs only to specify three parameters. The number of rows of the grid, the number of columns and the number sand grains to be added. Something of the form:[1]

```
class SandPile:
    def __init__(self, rows, cols, threshold=4):
        """Initialize a sandpile with the specified rows and columns."""
        # Grid Structure
        self.rows = rows
        self.cols = cols
        self.grid = np.zeros((rows, cols), dtype=int)
```

The object itself should be dynamic. That is there should be a function which enables the class object to be iterated through over time, updating the object with each iteration. Such a function would be called a 'simulate' or 'simulation' function and take the total number of iterations as input. Given the preceding discussion of parameters we are naturally led to define the number of time steps or iterations to be the number of sand grains. Thus one time step is considered to be the addition of a single sand grain.

2.1.2 Observables

Now we must specify what it is exactly that we are measuring during such a simulation and how such variables could be determined during the simulation. First recall that an avalanche event is any time step where a site is above the critical threshold and thus a sequence of topplings occur. Most of the observables we consider will be directly measuring some aspect of an avalanche event when one does indeed occur.

Topples. The number of topples $T \in \mathbb{N}_0$ is the number of times a toppling occurred during an avalanche. If the program has a toppling function this would simply just be a return of the number times the function was called.

Area. The number of unique sites $A \in \mathbb{N}_0$ that were toppled during an avalanche. This could be measured by storing all toppled sites into a set and then simply taking the cardinality of the set. Note this will be highly correlated with the number of topples and will only differ from the toppling observable if the sequence of topplings form cycles[2]. Moreover we see that for any avalanche it is always true that $A \leq T$.

[1] This is a verbatim environment not a figure so on latex I cannot label or caption it.

[2] 'Cycles' here means in the graph theory sense

Length. The length of an avalanche D is defined to be the furthest distance away from the epicenter (initial toppling site) that a site toppled. This could be determined by comparing the distance from the epicenter to the current site to be toppled with the previous maximum distance stored. One potential ambiguity is that there are many ways to measure distance. So given that our grid can be thought of as a subset of a 2 dimensional integer lattice $\mathbb{Z} \times \mathbb{Z}$ it seems most appropriate to use that L^1 metric also known as the taxi-cab metric. For two points $P = (m_1, m_2)$ and $Q = (n_1, n_2)$ on a lattice $\mathbb{Z} \times \mathbb{Z}$ the L^1 metric $d_1 : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ is defined to be the map[3]

$$d_1(P, Q) = \|P - Q\|_1 = |m_1 - n_1| + |m_2 - n_2| \tag{2.1}$$

This has the major benefit over the standard Euclidean metric (or L^2 metric) in that all values of D returned will be of type int and so performing analysis on such data does not require the fine tuning of bin widths, which otherwise would be the case if we were doing analysis on an array of stored floating point numbers.

Loss. This observable $L \in \mathbb{N}_0$ is the total amount of sand lost given an avalanche. Since the Abelian sandpile model structurally is a grid with open boundaries, piles that form on the edge of the grid could topple drop sand off the side of the grid. A way of determining how much sand was lost would be to check if the site being toppled is on an edge and if so increment the counter by 1, note however if our site lays at the intersection of two edges, a corner so to speak then we must increment the loss counter by 2 accordingly.

Mass. The total mass $M \in \mathbb{N}_0$ of the grid at each time step. This quantity by definition must be bounded by the capacity C of the grid that is $M \leq C$. Where for a grid of size $m \times n$ with threshold value h we have $C = mn(h-1)$. Such an observable can be calculated by simply summing the values of all sites of the grid.

Density. The density (short for mass density) ρ of the grid is simply the mass M of the grid divided by the area $A = m \times n$ for a grid of size $m \times n$. This quantity is not actually an entirely different observable instead it is just a rescaling of mass by the area of the grid. Nonetheless it is still quite a useful variable when doing analysis of statistically stationary state of the grid.

2.1.3 The Simulation Algorithm

The details of how simulation should work now need to be made explicit. The simulation will begin by a single sand grain randomly being added or "dropped" onto the grid. Next we would need to determine if any site on the grid is above the threshold value in which case that site would have be the initial site or "epicenter" of that avalanche. At first one might naively attempt to resolve such a problem by iterating over the grid. This does work however such an operation is of order $O(mn)$ (or $O(n^2)$ for square grids) so should be avoided if possible. A viable solution that runs in constant time is simply storing temporarily the site on which the sand was dropped and checking if that site is greater than the threshold. By construction it will then be the case that every other cell on the grid is of height less than 4. Since to be more than 4 without an avalanche a sand grain must have dropped on that site to make it critical. Therefore we check if the site we drop sand is above threshold after the grain has been added to it and not worry about other sites. This reasoning leads to dropsand() function which takes in a site and some number of grains and adds it to the site. If no site is selected than a random site is uniformly chosen and sand is dropped. That site is then returned as output.

Now we must consider what happens when a site is above threshold. So far we have defined an avalanche to be a sequence of topples so by that logic we might construct a function avalanche which does not itself topple the site but keeps track of the avalanche event as a whole. Instead avalanche calls another function 'topple' which then topples the site.

How the functions, 'avalanche' and 'topple' work is at the core of how the simulation performs overall. So choosing a method that is simple enough to implement but also has minimal time complexity is of utmost importance since the difference for large simulations could be on the order of hours. The two main styles of approach here are iterative or recursive. For an iterative approach avalanche might continuously iterate over the grid calling topple for each site. Which then topples the site until no sites remain. On the other hand a recursive approach would look something like avalanche calls the function topple which topples the site, adds a single grain to each site (assuming a threshold value of 4 here) and checks adjacent sites, calling itself again if any adjacent site is above a certain threshold.

The recursive approach seems to be well suited to the current model, and will yield a much better time complexity. The reason for the recursive approach being more optimal is a result of the recursive algorithm utilizing an important fact that the iterative approach does not. Topples are local. This means we know that most of the grid already does not contain sites above a critical threshold so we should only care about the sites adjacent to the ones we have toppled.

[3]Such a metric is obviously well defined on more general space as well for example $\mathbb{R}$

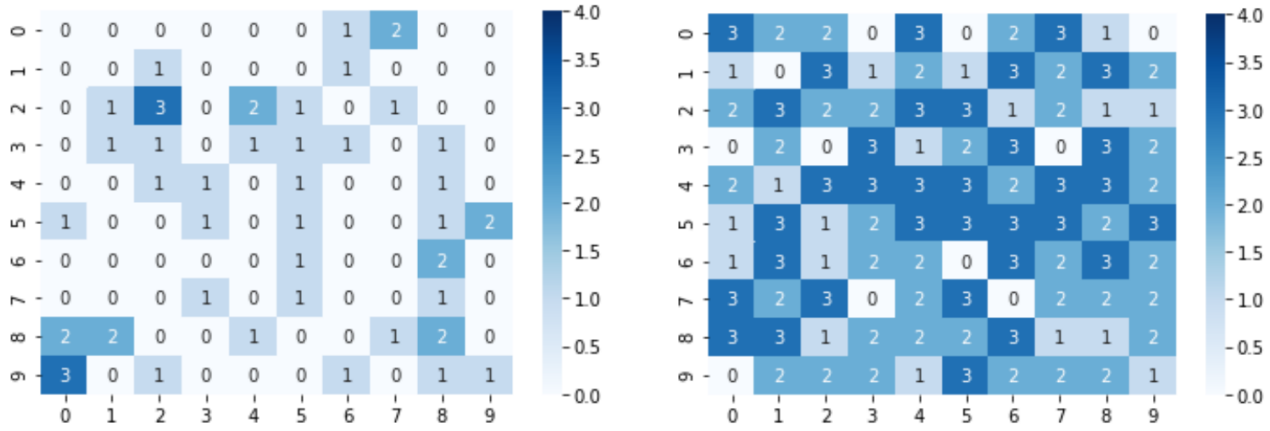
That is not to say that such an iterative approach is without its merits. As we will see later, a recursive algorithm will not be able to determine the lifetime of an avalanche event and so that observable cannot be measured in the recursive model.

Lastly it is worth mentioning that such a recursive method heavily relies on the fact that the toppling of the sandpile is abelian. For the mathematical proof of the abelian nature of the BTW-model see A.2.1.

2.1.4 Implementation

The Abelian Sandpile model was successfully implemented in the manner described above, using a class for the sandpile model along with functions such as 'simulate', 'avalanche','drop sand' and 'topple'. In short a recursive approach was used for the toppling with structure as seen on the flow chart (Next page).

In order to check that the model was working as it should. A way of seeing the grid as the dynamics were being simulated was necessary. The 'visualisation' option was thus added as a parameter to the simulation function. When the user sets 'visualisation' to true an animation of the simulation appears on screen. In the animation each site is labelled with its value and sites of the same value have the same colour. Figure 2.1 demonstrates this and shows the simulation of a 10×10 grid at instances $t = 50$ and $t = 500$.



(a) Sandpile at $t = 50$

(b) Sandpile at $t = 500$

Figure 2.1: Animation of the sandpile model

The visualisation option though has its limits. Upon testing the visualization feature for simulations with large grid sizes it seems as though the numbers get too difficult read and the render time gets significantly longer. For that reason an if statement was implemented to stop the user from choosing visualization for grids with an area larger than 400 sites. Such a cut off value was chosen through trial and error.

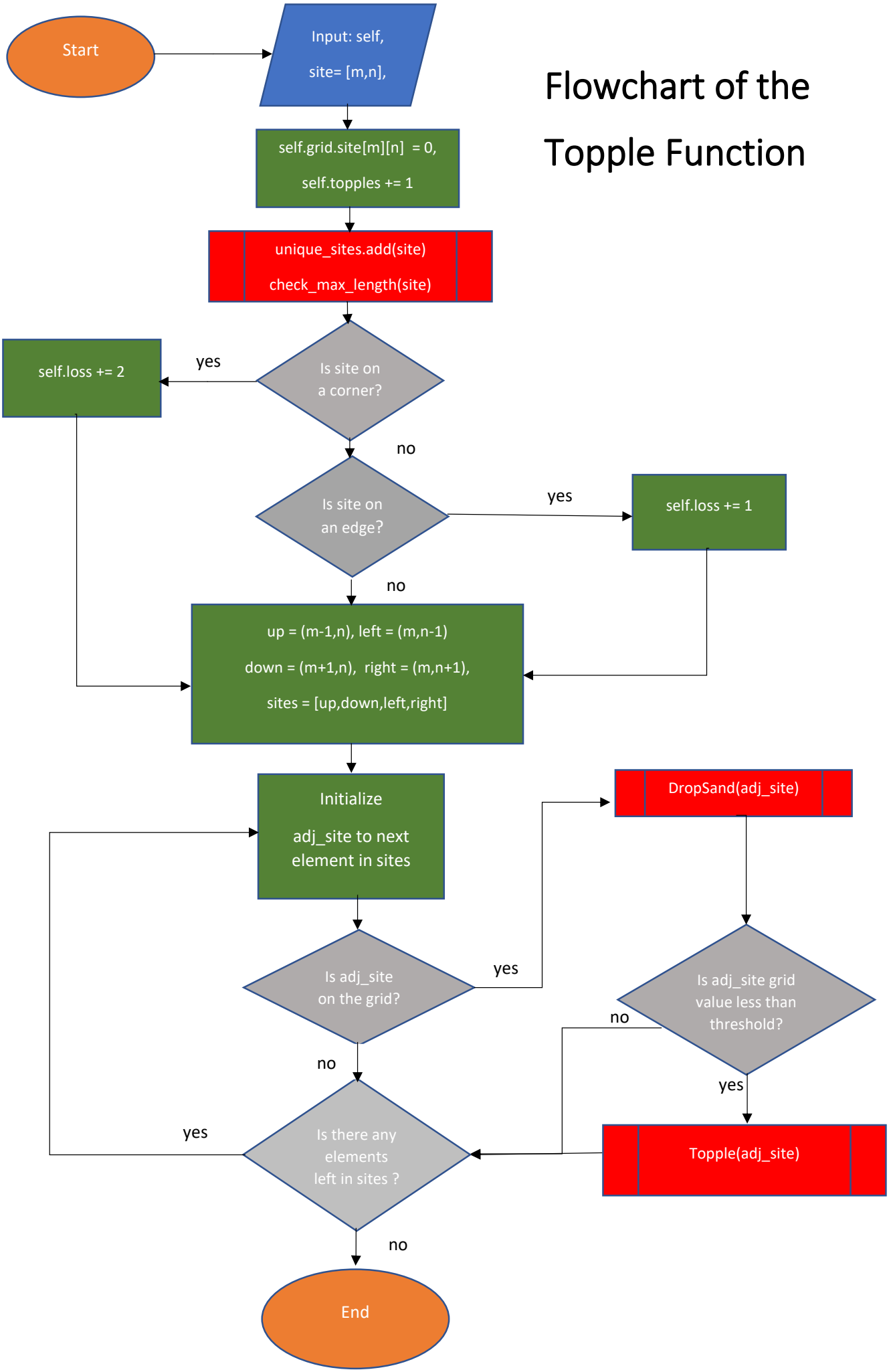
Some more thought was given to the user experience as for simulations with long run times one would find themselves not knowing if the simulation was progressing or if an error had occurred and the program was stuck. Therefore a progress update function was added to the class where a user could parse in a parameter 'notify every' of type float into the simulation between 0 and 100 and every time the simulation reached a multiple of that value the user would be notified of the progress of the simulation.

```
simulation progress: 0.00% completed
simulation progress: 1.50% completed
simulation progress: 3.00% completed
simulation progress: 4.50% completed
simulation progress: 6.00% completed
```

Figure 2.2: Progress Bar with notify increment set at 1.5%

Lastly in order to make the data collection more streamlined an update function was constructed called at the end of each sand drop. It has no parameters other than self and it simply appends all the avalanche data into their respective data storage arrays in the sandpile class.

Flowchart of the Topple Function



.

2.1.5 Sandpile Analytics

One of the core aims of building the simulation is to verify that observables for this system follow power laws in the critical state. Thus the data we require must be of the observables only whilst the sandpile is in the critical state. Otherwise our data will contain outliers from the initial state of the grid and this would in turn create error in our calculation of basic statistical quantities, such as the mean, median etc. Therefore a truncation or lower cut off value must be determined for the time step such that all data collected after the cut off is in the critical state.

Since we want to truncate all the data at the same index, concatenating each array of observable data into a single large data frame was performed

	mass	density	topples	area	length	loss
0	0	0.0000	0	0	0.0	0
1	1	0.0016	0	0	0.0	0
2	2	0.0032	0	0	0.0	0
3	3	0.0048	0	0	0.0	0
4	4	0.0064	0	0	0.0	0

(a) Head of the Dataframe

	mass	density	topples	area	length	loss
9996	1308	2.0928	0	0	0.0	0
9997	1309	2.0944	0	0	0.0	0
9998	1310	2.0960	0	0	0.0	0
9999	1311	2.0976	0	0	0.0	0
10000	1312	2.0992	21	21	6.0	0

(b) Tail of the Dataframe

Figure 2.3: 10000 sanddrops on a 25×25 grid

Now the objective was to determine the cut off. That is finding when the simulation has reached the critical state. An approach that yielded inconsistent results was to analyse the relative maximum and minimums of the mass time data. Since the total mass will be increasing on average and not fluctuating when the grid is in the initial state. However when the grid is in SOC the total mass should be oscillating around a statistically stationary mean. The flaw in that approach was that even though the total mass does increase overall as the system is in the initial state, locally it still fluctuates and so the functions implemented cannot distinguish between the local fluctuations in the initial state and fluctuations in the critical state **consistently**.

Luckily[4] a cheap and perhaps over-simplistic, but very effective method of determining the cut off was soon realized. It comes from the following highschool level of analysis:
Recall that the total mass M of the grid is bounded by the capacity C of the grid, that is equation 2.2

$$M \leq C = mn(h-1) \text{ where } h \text{ is the threshold.} \tag{2.2}$$

Since the mass can never exceed this value. The number of sand grains (or time steps) required to fill the grid to capacity seems to be at a time step where the grid is always in the critical state. Therefore using this value we see that the data frame can be correctly sliced to have only data that is in the critical state.

Since the data will be sliced at the index corresponding to the capacity one might then wonder what if the user inputs capacity+1 sand grains. Performing statistics tend to be useless when N is small, so after implementing the cut off we imposed that for the statistics to be done on the grid the user must specify a number of sand grains (time steps) larger than $1.5C$ that way we ensure there is always some data be collected.[5]

For the displaying and processing of data standard libraries such as MatPlotLib and Seaborn were used. For doing regressions numpy.polyfit() was used. As power laws have the form

$$f(x) \sim x^m \text{ where } m \text{ is usually negative.} \tag{2.3}$$

we plot our observables on a log-log plot, and using numpy.polyfit() to do a regression. Since the coefficient from the degree one term of the regression corresponds to the degree of the power law.

$$\log(f(x)) \sim \log(x^m) = m\log(x) \tag{2.4}$$

[4] After much pain.

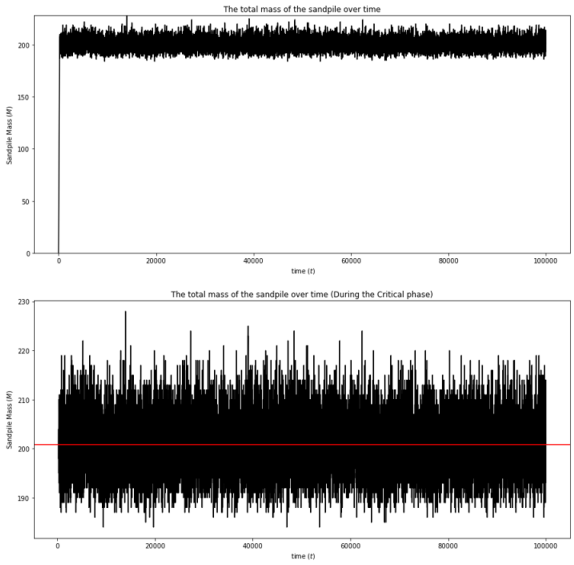
[5] Since this value should is a number of sand grains it cannot be a float, so what is really meant here is 1.5*C rounded to the nearest integer.

2.2 Results and Discussion

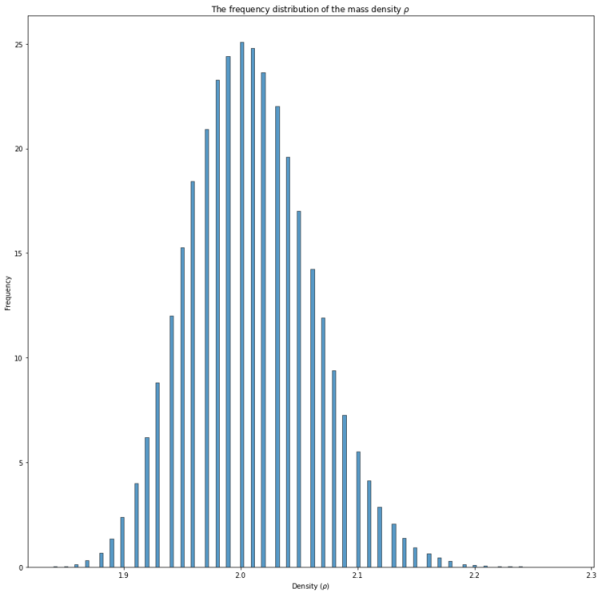
2.2.1 Mass and Density

In the simulation for a 10 × 10 grid with 100000 sand grains and a threshold of 4 we can see in figure 2.4a that the sandpile has a very distinct initial state where the mass is steadily increasing. Once the total mass of the grid is about to reach the grids capacity avalanche events become larger and generate huge losses leading to an overall decrease in the grids total mass. The equilibrium between the increasing total mass and total loss per iteration is seen to be non-stationary and instead oscillates about a value for which could be considered as the average height or mass density of the grid. This statistically stationary point can be seen in fig 2.4a as the red line on the mass vs time plot of the model in the critical state.

Moreover we found that the mass density observable follows Gaussian statistics as seen in Fig 2.4b with a mean of 2.01 grains per site and a standard deviation of 0.05. Recall that in the abelian sandpile model section 1.2 we estimated that the average height/mass density of the grid would have value 2.12 grains per site. Such a crude estimate was deduced by taking the arithmetic mean of an upper and lower bound has led to a surprisingly accurate prediction having coming within 5.5% of the true value.



(a) The total mass of the sand pile over time



(b) Histogram of the density in the critical state

Figure 2.4: 10 × 10 grid with 10000 time steps

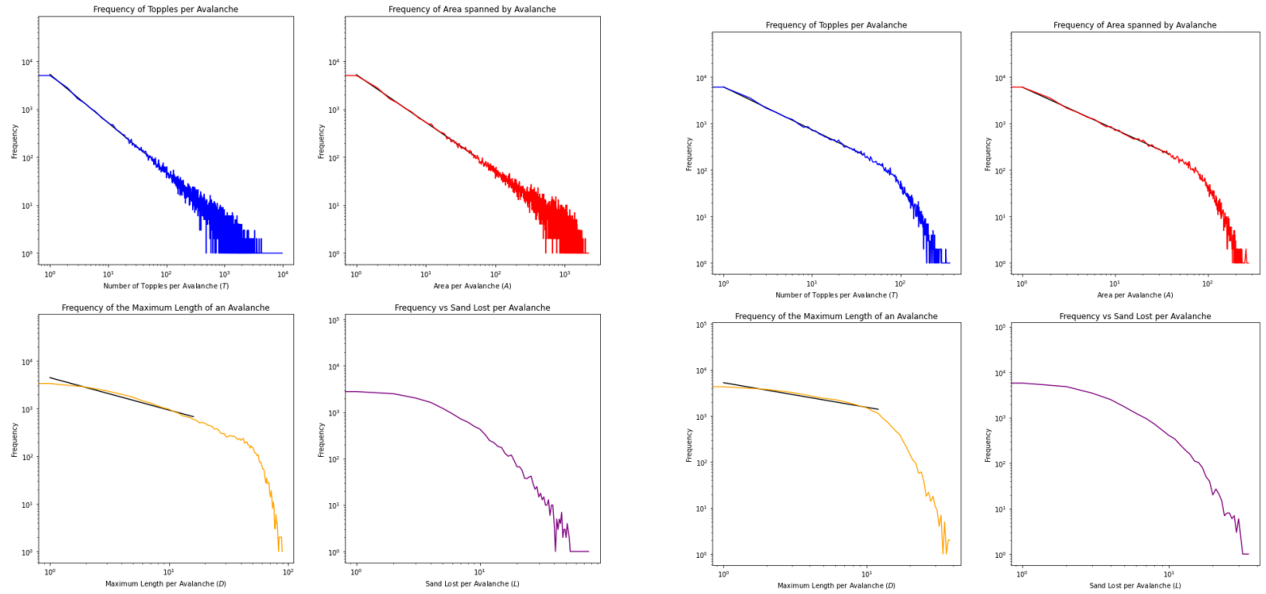
The abelian sand pile model also has the feature that it is ergodic since many different simulations all produce the same statistical distributions in the critical state. Therefore sampling a model that is in the critical state many times is the same as sampling one model numerous times as it evolves within the critical state.

2.2.2 Power Laws and Correlations

A square 50 × 50 grid with 10000 sand grains was simulated and clear power laws were observed as seen in 2.5a for three out of the four observables. For comparison a rectangular grid of the same area 10 × 250 was also then simulated with output as seen in 2.5b. Immediately we observe that loss is not a power law as it is convex throughout its entire range of values. This is makes sense since as previously discussed in section 1.2 the presence of the loss observable being non-zero is a result of the grid being finite and thus loss is a function of the scale of the grid and so is not scale invariant. More generally we see that the remaining observables all follow power laws (up to suitable truncation) with degree close to -1. In accordance with the theory discussed in section 1.1

Observed degree of power law			
Grid size	Topples	Area	Length
50 x 50	-1.00	-0.99	-0.67
10 x 250	-0.93	-0.91	-0.77

Table 2.1: Simulation results 100000 time steps.



(a) 50 × 50 grid 100000 time steps

(b) 10 × 250 grid 100000 time steps

Figure 2.5: The frequency distribution of each avalanche observable

Plotting the trend line also shows that our cut off for the truncation is correct with the noise featuring at the right of each plot avoided. Such noise is due to the grid being finite and would not be seen on an infinite grid.

We observe the length observable has a degree furthest away from -1. This could be due to our choice of how length is defined and suggests perhaps one should use a new measure of avalanche distance such as taking the diameter of an avalanche if this experiment were to be repeated in the future. [6]

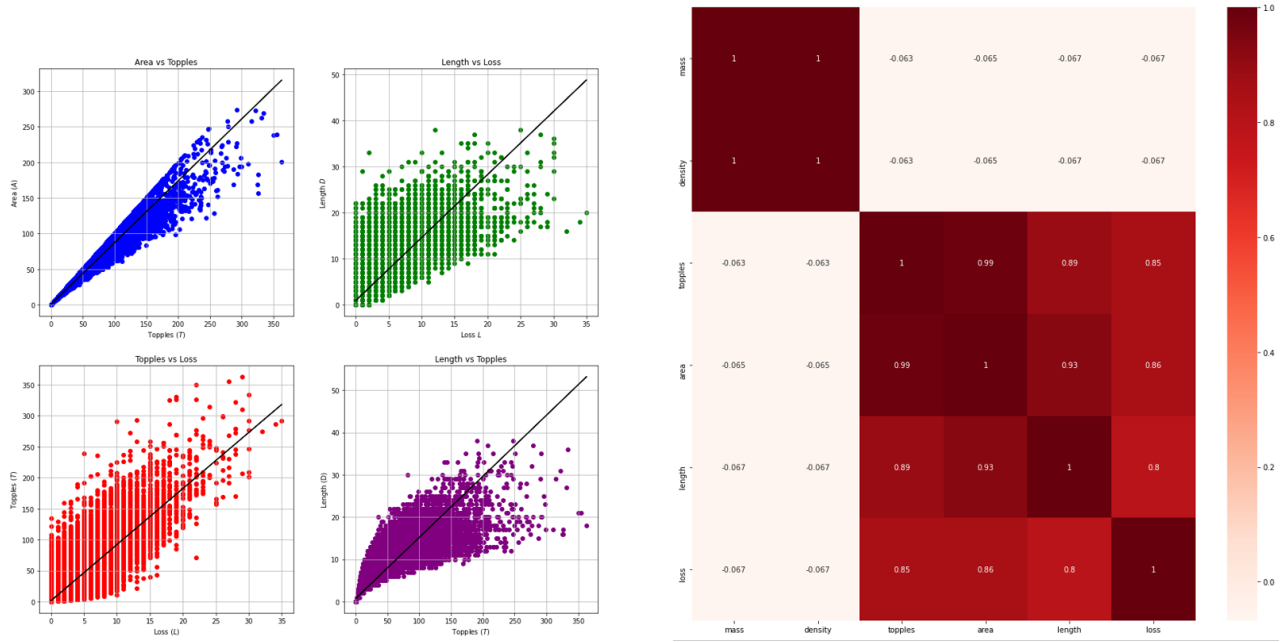
Lastly we make mention of the differing results as it corresponds to the shape of the grid. Firstly notice that the power law behaviour seems to be less apparent for the rectangular grid as we can see convexity occurring for large values of each observable. Such a behaviour is likely due to the fact that the rectangular grid has a perimeter of 512 sites approximately 5 times larger than the perimeter of the square grid (100 sites) of the same area. Therefore we have found that a lack of symmetry of the grid implies the grid is less scale invariant and this is reflected in the greater levels of convexity found in the power law data.

Now analyzing the output of the correlation matrix in 2.6b we see that all avalanche observables are correlated and the scale invariant observables are strongly correlated. Quantitatively that means they have a correlation of about 0.9 or above. The avalanche variables that were least correlated were loss and the length of the avalanche. This is could be due to the fact that we are considering here maximum length that is the maximum distance achieved by one toppling path from the epicenter of the avalanche. If we instead considered the average distance of each of the toppling paths from the epicenter we may find this observable has an even stronger correlation with loss since it gives more information about the spread of the avalanche as whole. One foreseeable limitation is that if we only consider the maximum length of one toppling path then this could itself be an outlier value and therefore not be a reliable source for the 'spread' of an avalanche.

We also see in 2.6 that the observables area and topples are almost perfectly correlated. The reason for this becomes apparent when we consider the definitions of both observables and then realise that they are only are different when an avalanche event (viewed as a graph) contains cycles. As area only corresponds to the total number of unique toppling sites. So roughly speaking for every cycle an avalanche event contains we will see a 1 count difference between area and topples. Contemplating this further we now have realised that what we have found is a measure of the prevalence of cycles in our avalanches represented by the deviation of of area-toppling correlation from 1.00.

Lastly in the scatterplot for length and topples we see that the line of best fit will not give a good extrapolation of the data as the data seems to be trending towards the right of the plot. Qualitatively we can make sense of this as the fact for a finite grid the maximum distance of an avalanche event is tightly capped by the grid size whereas the numbers of topples is not.

[6] Here we mean diameter in the graph theory sense of the word.



(a) Scatter Plots showing the correlated observables

(b) Correlations Heatmap

Figure 2.6: Correlations between variables for the 50 × 50 grid

2.3 Limitations of the Recursive Algorithm

Despite the previous discussion of the performance benefits (see table 2.2) of a recursive approach to the implementation of the BTW model, it is likely not suitable for either of the extension models which are more complex. Ultimately the reasons for having to abandon recursion are two-fold. Firstly if a recursive algorithm is used then measuring the avalanche observable **lifetime** (discussed at the start of the next chapter) is not feasible since in a recursive algorithm we are not toppling sites according to the iteration in which they appear.

Secondly, recall from the end of subsection 2.1.3 that using a recursive model hinges on the assumptions that there cannot exist multiple sites on a given time step which need to be toppled and that toppling is abelian. Such an assumption is valid in the case of abelian sand piles but when we consider an earthquake model where the stress of every site is incremented at once this is no longer true.

Such an algorithm also adds an unnecessary layer of complexity when trying to make modifications to the sand pile such as adding modifying the distribution of sand and adding boundary conditions.

Performance of Recursive Sandpiles Algorithm for varying grid size					
Grid size	10 x 10	25 x 25	50 x 50	75 x 75	100 x 100
100000 Timesteps	5.08s	21.16s	71.78s	146.08s	232.97s
500000 Timesteps	25.36s	102.55s	369.93s	793.43s	1332.84s

Table 2.2: Recursive Sandpile Model Runtime Data

Chapter 3

Extendeing the BTW Model

3.1 Methodology

3.1.1 An Iterative Approach

Following the discussion of the limitations of a recursive approach in the previous section a new sandpile model was made with all the previous features retained but now with the a toppling method that scans the grids for sites to topple. Such a toppling method was implemented using the NumPy function np.where() to return the indices of all sites above the threshold value. A while loop would then iterate through this list and call the function topple for each site on the grid. Once all sites are toppled the grid is then scanned again for sites above the threshold value.
Lifetime. Define the lifetime $\lambda \in \mathbb{N}_0$ to be the number of iterations taken for the avalanche event to end. Such an observable can be determined by keeping track of the number of times np.where() is called in an avalanche event.

In the extended BTW model the topple function is different from the one seen in the flowchart in chapter 2. This one works by simply toppling the given site, calling the function 'gets_adjacent_sites()' which returns all adjacent sites and then calling the function distribute_sand() which decides how the sand is distributed.

3.1.2 Redistribution Methods: Uniform and Gradient Toppling

One of the parameters we would like to able to adjust is the threshold amount for a topple to occur. However with the previous algorithm changing this number to a value that is not a multiple of four would lead to errors since the redistribution of sand during a topple was such that each adjacent site would receive a quarter of the threshold amount of sand grains.

As a way of allowing the threshold parameter to be any positive integer. The decision was made to consider redistributing each sand grain individually according to a uniform probability distribution on the adjacent sites. Succinctly put, if a topple was now to occur each adjacent site has a 1 in 4 chance of receiving that grain.

Such a redistribution method works for thresholds with any arbitrary positive integer value. However such a way of redistributing sand seems far fetched from reality (If three adjacent sites are at height 100 and the other site has height 3 then our physical intuition tells us that toppling the pile should probably result in a disproportionate amount of sand going to the site of least height).

Such a redistribution method is known as gradient toppling and provides a much more realistic model of how actual redistribution would occur in a system like a sandpile. In our simulation this works by assigning each adjacent site a probability corresponding to its height difference with the threshold site. That is the larger the height difference the more likely each sand grain would be distributed to the site. To determine the form of this probability distribution mathematically consider the following. let $n = 1, .., 4$ be the index of an adjacent site to the site being toppled with value $p_t(n)$ at time t. Then the difference in height between an adjacent site and the toppled site is given as $d_n = h - p_t(n)$ where h denotes the threshold value of the grid. Suppose D is the sum of all the height differences between the toppled site and adjacent sites. Then we have that the probability $P_1(n)$ of site n with height d_n being given a single grain is given by

$$P_1(n) = \frac{d_n}{D} = \frac{h - p_t(n)}{\sum_{n=1}^{4}(h - p_t(n))} = \frac{h - p_t(n)}{4h - \sum_{n=1}^{4} p_t(n))} \tag{3.1}$$

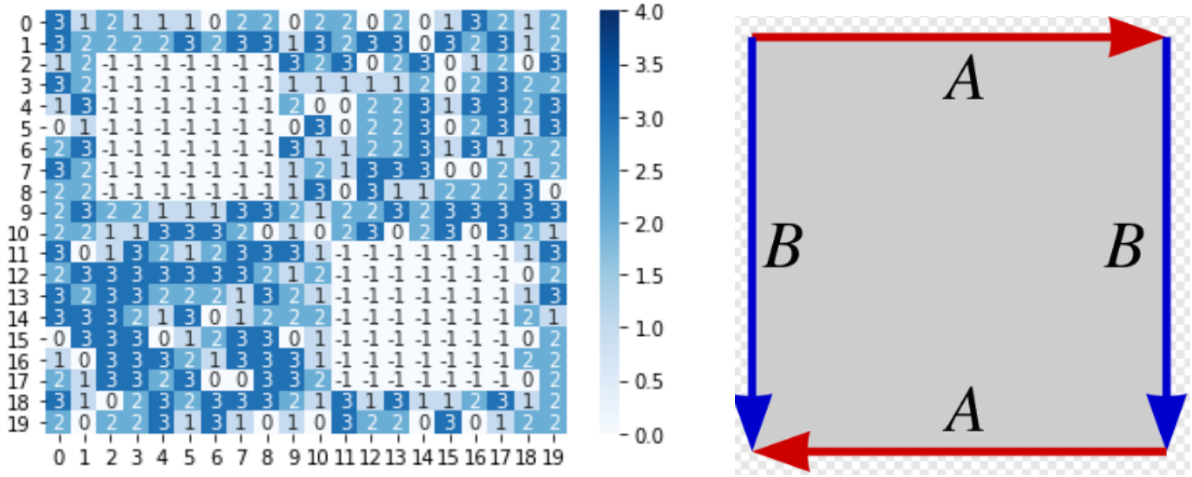
Equation 3.1 provides us with an easy way of implementing the probability distribution for gradient toppling into the simulation as the probabilities are in terms of variables we already have stored.

3.1.3 Turning the grid into a Cylinder, Möbius Band, Torus or Klein Bottle.

The one thing that has so far remained constant throughout all our simulations is the fact that our grid is rectangular. We now wish to change this by gluing edges and creating holes in our grid.

Since the grid is a rectangular polygon, we can 'glue' a single pair of opposite edges to get a cylinder. If we instead reverse the boundary orientations before gluing we get a Möbius band. If we glue both pairs of edges we get a Torus and if we reverse the orientation of an edge before we glue one of the pairs then we get the Klein bottle. Therefore in implementing these exotic grids into the code we only really needed to change the boundary conditions of the rectangular grid to agree with gluings just described.

There is one issue however that arises when exploring the klein bottle and torus as grids and that is there can be no loss on such objects since they are manifolds without boundary. To compensate for this the ability to puncture the grid with a hole was added and the code was adjusted so that a simulation on a Klein bottle or torus must have at least one hole to execute without raising an error. The hole in the grid is represented by having values of -1 as seen in 3.1a



(a) The sandpile model reaching the critical state on a Klein bottle with two punctures.

(b) The fundamental polygon of a Klein bottle showing the orientations of the gluings.

Figure 3.1: The abelian sandpile model with the Klein bottle topology.

3.2 Results and Discussion

3.2.1 Toppling Method

A 50×50 rectangular grid with 100000 sand drops was simulated twice varying the toppling method. The results are shown in table 3.1. We see that both methods give approximately the same value for the degree of the power laws with deviations most likely due to the inherent variance in the simulations. Comparing the run time with table 2.2 we see that the addition of randomness in distributing each grain has added a significant amount of length to the time taken for the simulation to complete. With the gradient toppling taking almost double the run time as compared with the uniform toppling method. This is to be expected as determining the probability distribution for the gradient toppling requires additional algorithms. Due to the large discrepancy in run time and lack of deviation in the overall results we justify retaining the uniform toppling method and making it serve as the default method for our extended model despite it theoretically being less realistic. The gradient toppling method can still be used if desired by changing the toppling method argument to 'gradient'.

	Topples	Area	Length	Lifetime	Runtime
Uniform	-1.09	-1.09	-0.85	-1.13	206.58s
Gradient	-1.11	-1.10	-0.88	-1.11	371.97s

Table 3.1: Simulation results for a 50 x 50 grid with 100000 time steps and varying toppling method.

	Observed Degree of Power Law				Extreme Value Statistics	
Topology	Topples	Area	Length	Lifetime	Largest Avalanche (Topples)	Longest Avalanche (Lifetime)
Rectangle	-1.09	-1.09	-0.85	-1.13	14153	650
Cylinder	-1.08	-1.07	-0.94	-1.10	30610	1104
Möbius Band	-1.09	-1.07	-0.97	-1.12	32585	1129
Torus (1 puncture)	-1.08	-1.06	-1.09	-1.09	39579	1670
Klein Bottle (1 puncture)	-1.10	-1.08	-1.08	-1.10	41706	1424

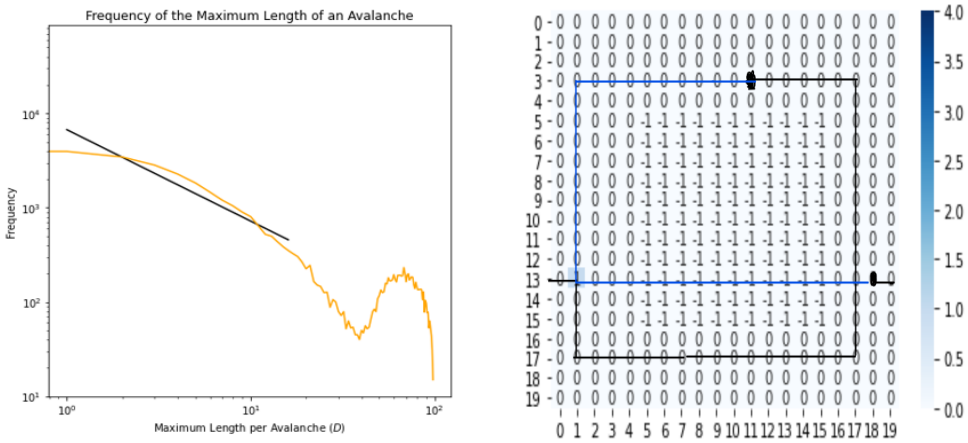
Table 3.2: Simulation results for a 50 x 50 grid with 100000 time steps and varying grid topology.

3.2.2 Lifetime and varying the grid topology

The observable lifetime was found to consistently follow a power law distribution as shown in Fig 3.2a similar to all other avalanche observables (excluding loss). Thus using the same truncation for lifetime as we have for other observables was found to appropriate.

Analysing the results in table 3.2 we see that changing the topology has little effect on the degrees of each of the observables however it has a significant impact on the extreme values statistics for avalanche events. Specifically we see that for a cylinder and Möbius band the largest avalanche and longest avalanche lifetime are double that of the standard rectangular grid. This result makes sense given we have eliminated the boundary of 2 opposite edges on the grid thus halving the number of sites where an avalanche can end. The torus and Klein Bottle both have extreme values statistics a third larger then the cylinder and Möbius band. This is likely due to both objects having no exterior boundaries but instead a punctured 10 x 10 hole in the center so that loss can occur. We observe that the perimeter of this hole is 40 sites whereas the for the cylinder and Möbius band the total perimeter of the unglued edges is 100. Therefore if we suppose an avalanche event can topple as far as it likes. Then the probability such an avalanche hits an upper or lower edge is much higher in the case of a cylinder or Möbius band.

Lastly we briefly discuss the limitations of measuring distance with our taxi-cab metric on non-standard grids. Figure 3.2 shows if the avalanche event crosses a boundary then our current method of determining distance will fail to detect this and instead ascribes a large distance to the avalanche when it did not have one. This error manifests itself in a local maximum as seen in 3.2a. Such an issue when dealing with simulations that involve periodic boundary conditions is known and arises in many different contexts. Either a transformation of coordinates or a different approach to the distance observable altogether would be required if this experiment were to be repeated.



(a) Length Observable for a torus.

(b) Cases of the taxicab metric failing. Path-Black, Metric-Blue.

Figure 3.2: The length observable for non-standard grids.

Chapter 4

Extension: Earthquake Modelling

4.1 Background

4.1.1 What causes an Earthquake?

The Earth's crust also known as the lithosphere is made up of 15-20 tectonic plates which fit together in a jig-saw puzzle like way and sit above hot magma. The meeting point of two tectonic plates is called a fault (short for fault line). At a fault the border of a single plate experiences accumulating stresses due to the ongoing slow collisions caused by both plates slowly moving against each other. When the plate, which is stressed, sheared and/or compressed is deformed so strongly that the deformation exceeds a certain limit (which varies from place to place) rupture occurs and an earthquake follows. The plate then rebounds and the accumulated strain energy is radiated in the form of deadly seismic waves.

Analysing the description above the parallels between Earthquake phenomena and the abelian sandpile model start to become more transparent.

4.1.2 The Gutenberg-Richter Law

In 1956 Charles Richter and Beto Gutenberg published a paper [3] where they proposed a relationship between Earthquake magnitude and frequency. Since then the law has become a fundamental result in the field seismology and is called the Gutenberg-Richter Law. It states that the relationship between the magnitude and total number of earthquakes in any given region and time period of at least that magnitude is modelled by the equation

$$\log_{10}(N) = a - bM \text{ where } a, b \text{ are constants} \tag{4.1}$$

where M is magnitude and N is the number of events having magnitude greater than M. Visually we can see the power law relationship in 4.1. Now if we introduce the variable N_{Total} for the total number of Earthquakes we can re-express the G-R law in the following form.

$$N = N_{\text{Total}} 10^{-bM} \text{ where } N_{\text{Total}} = 10^{a} \tag{4.2}$$

This form of the G-R law is much more useful if we are trying to determine the b value from earthquake data.

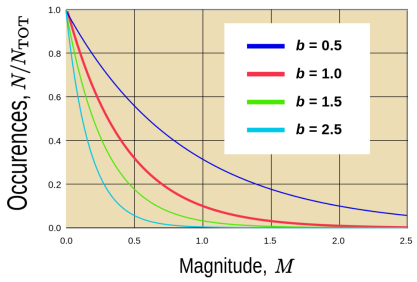


Figure 4.1: Gutenberg-Richter Law for different b values

4.2 Methodology

4.2.1 Implementing the Earthquake Model

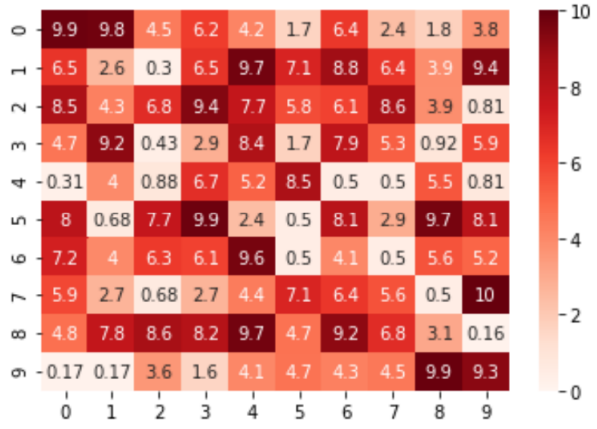
With the relevant background understood we proceed with creating the Earthquake model. The Earthquake model will differ from the abelian sandpile in a few major ways. The first is that every site on the lattice will now be incremented by the same amount at each time step. Next we have that the conservation of sand or stress energy in the context of earthquakes now depends on a parameter α which ranges between 0 and 0.25. Lastly the grid is now initialized with random floats of small values relative to the threshold.

We began by reusing most of the functionality of our extended sandpile model code. Using np.random.uniform() to initialize the grid with small floats. The function add_stress() was created to simply increment the entire grid. In order to not over complicate things we defined 1 time step in our model to be 1 call of the add_stress() function, so 100 time steps is 100 increments of the entire grid. The dissipation parameter α was added as a class variable and included in the sand distribution function. Lastly the statistics for how loss is calculated was adjusted for to include the dissipation due to alpha.[1] since this avoided artificial run time issues and rounding errors.

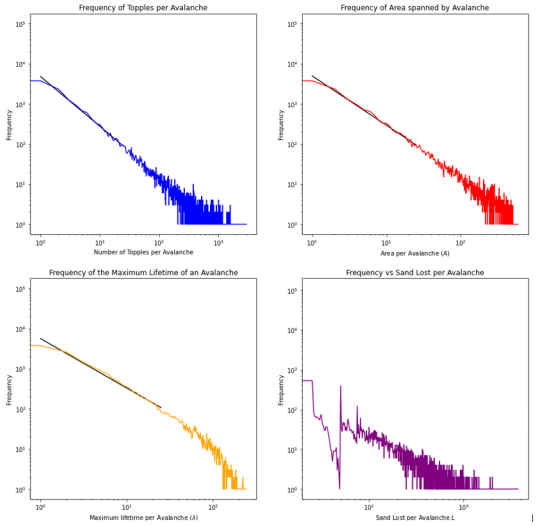
4.2.2 Modelling

Data from 2019 to 2022 of all earthquakes in the US was scraped from the United States Geological Survey website and imported as a CSV file into our earthquake model. This data was then converted into a pandas data frame in order to be more easily managed. Subsequently the data was plotted and the b-value was determined using np.polyfit() to be exactly 1.00.

Next we used our earthquake model in conjunction with the data to predict the value α for earthquakes in the United States during 2019-2022. Given we do not a have a magnitude observable we first defined that an earthquake of magnitude 1 corresponds to an avalanche event with precisely 10 topples. Then by considering a large grid of dimensions 75 $\times$75 we slowly varied the parameter α. Noting that when $\alpha = 0.25$ we get a b-value much below 1.00. So each trial we incremented α by 0.02 until we found a corresponding b-value above 1.00 at which point we the use the midpoint for the next trial, obtaining the results in the next section.



(a) A screenshot of the Earthquake model with threshold 10 and increment 0.01 in the critical state. ($\alpha = 0.24$)



(b) Observed power laws in the Earthquake model.

Figure 4.2

[1]It should be noted that instead of using very tiny floats on the order of 10^{-5} and single digit threshold value, as was done in the original instantiation of the model. We used large threshold values and our grid was initialised with floats on the order 10^{-2}

Trial	1	2	3	4	5	6	7	8
α	0.2400	0.2200	0.2000	0.1800	0.1600	0.1700	0.1750	0.1725
b	0.1600	0.3400	0.6300	0.8900	1.200	0.9600	0.9500	1.020

Table 4.1: Earthquake modelling data for a 75 $\times$75 grid with threshold value 100 and stress increment 0.02.

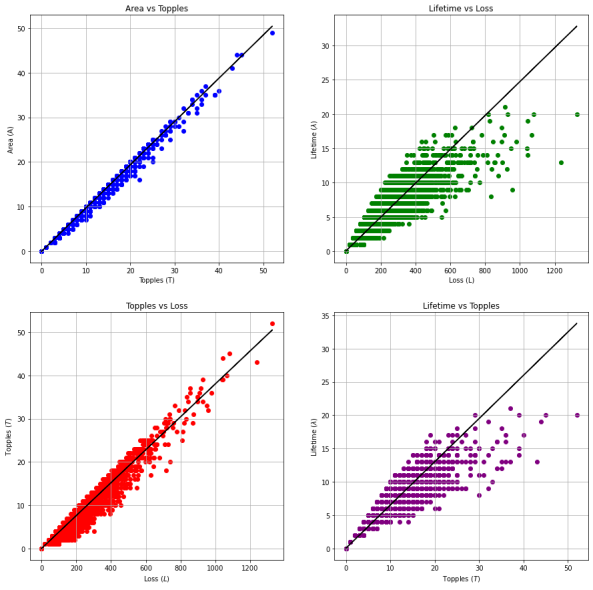
4.3 Results and Discussion

Firstly we note that the Earthquake model was shown to achieve a critical state and thus exhibit the property self organized criticality. We also observed (see 4.2b) power laws in the earthquakes critical state similar to the abelian sandpile model with all avalanche observable having a degree close to -1.00. A noticeable difference between the earthquake model and the sandpile model is that the observable loss has a very strange distribution. This is seemingly due to the dissipation's between topples increasing the total sand loss when the number of topples is high. Such an explanation is also supported by figure 4.3a where we see loss and number of topples now have a very strong correlation.

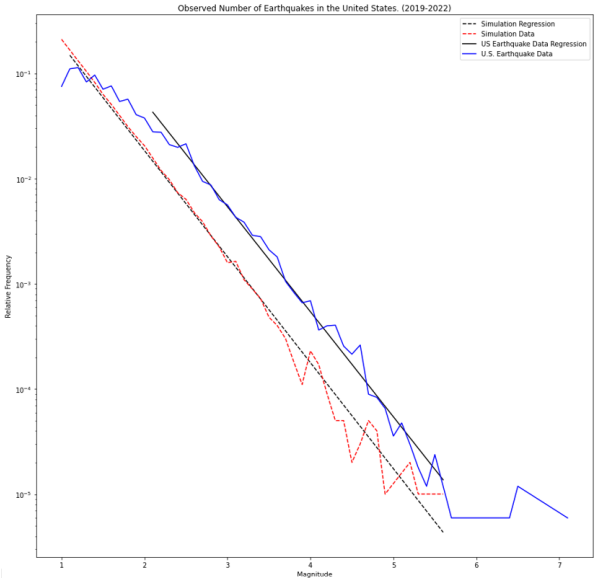
Table 4.1 reflects the approach discussed at the end of the previous section as we can see an small incremental variation of alpha occurring with each trial performed until trial 5 at $\alpha = 0.1600$ where a lower bound is found for α. We see within the table that 0.1725 returns a value of b not between either of the b values corresponding to 0.1700 or 0.1750. The reason for this is that the variance inherent to the simulation itself is having an effect when considering such fine values. For this reason after the 8th trial it was decided to take $\alpha = 0.1725$ since searching the parameter space in this region for a better value would be difficult given the variance in the simulation.

Therefore our model suggests that the percentage off energy dissipated per site in the U.S. Earthquake data is 0.2500 - 0.1725 = 0.0775 = 7.75% which seems to be a very reasonable estimate.

Lastly we would like to briefly the discuss the limitations of the approach of the model used. Specifically by slowly varying alpha we were able to find an extrema close to our starting point of $\alpha = 0.25$ however this does not exclude the possibility that there could be other potentially more optimal extrema that we have neglected by not computing α for an evenly spaced range of values across its domain. Another potential limitation is that perhaps the 75×75 grid size used was too small, as we can clearly see in figure 4.2 despite having the same value for the gradient, the area under the curve of our model is smaller than that of the data. If this experiment were to be repeated, optimizing to try and find the correct grid size which is not too large or small to align with the data should be done first before varying the parameter α.



(a) Correlations between avalanche observables ($\alpha = 0.20$)



(b) Earthquake model vs U.S. Earthquake for $\alpha = 0.1725$

Figure 4.3

Chapter 5

Conclusion

We successfully implemented the abelian sandpile model in python 3.9. using a recursive algorithm. The phenomena of self organized criticality was then explored and observables relating to avalanches were all found to follow power laws except for loss. We uncovered that loss was an artifact of finite grid size and thus proportional to the grid size and so not scale invariant.

The model was then extended to include more dynamic features, such as a more realistic gradient toppling method as well the ability to change the topology of the grid. Different methods of gradient toppling were found to yield the same power law statistics but have vastly differing run-times. Additionally we found that the changing the topology of the grid did not change the power laws corresponding to area and topples but were instead reflected in drastic changes to the extreme value statistics of avalanche events. The limitations of measuring avalanche distances in curved geometries was also encountered and discussed.

Finally an earthquake model was built using the extended sandpile model. The model was shown to observe self organized criticality with the power laws governing the distribution of avalanche observables (excluding loss). In such a model the inclusion of non conservative topples revealed a new stronger correlation between topples and loss that previously was not seen.

The data of all Earthquakes in the United States between 2019 and 2022 was modelled to show the Gutenberg-Richter law empirically. The data had b-value exactly 1.00. Subsequently we used our earthquake model in accordance with the data to determine the value of alpha. The value of $\alpha = 0.1275$ turned was found to give the best fit resulting in the value for the stress energy dissipation of Earthquakes to be 7.75% per site for the given data.

If this computational report were to be repeated in the future, all the limitations discussed at the end of each section should be taken into account. Specifically we emphasize the need to do a better job when it comes to the measurement of avalanche distance/length. One potential avenue for extended was done here would be to create a 3 dimensional abelian sandpile model. Then of course modifying the topology of that grid in a similar way to what was done here in chapter 2 to see if self organization still occurs. "Abelian sand piles on the 3-torus".

Appendix A

A mathematical approach to the BTW-Model

A.1 Definitions and Fundamental Results

Consider the lattice of sandpiles in dimension d to be a subset $L \subset \mathbb{Z}^d$. For any site $\mathbf{x} \in L$ the number of each site of the lattice is given by the function $p : L \subset \mathbb{Z}^d \to \mathbb{N}_0$. A **stable configuration** corresponds to when $p(\mathbf{x}) < k$ for all $\mathbf{x} \in L$ whilst an unstable configuration has at least one site $\mathbf{x} \in L$ such that $p(\mathbf{x}) > k$ where k defines the **stability threshold**. For an unstable configuration we require a way to describe the toppling, thus we introduce the toppling matrix $\Delta(\mathbf{x}, \mathbf{y})$ which redistributes the sand grains from pile $\mathbf{x}$ to $\mathbf{y}$ when $\mathbf{x} \neq \mathbf{y}$.

The toppling matrix must satisfy the following conditions:

1. **Symmetry**, such a matrix should have a non reflexive symmetry.

$$\forall \mathbf{x}, \mathbf{y} \in L \text{ where } \ \mathbf{x} \neq \mathbf{y}, \ \ \Delta(\mathbf{x}, \mathbf{y}) = \Delta(\mathbf{y}, \mathbf{x}) \geq 0. \tag{A.1}$$

2. **Self Loss**, A toppling at site $\mathbf{x}$ implies a net loss of sand at site $\mathbf{x}$.

$$\forall \ \mathbf{x} \in L \ \ \Delta(\mathbf{x}, \mathbf{x}) < 0. \tag{A.2}$$

3. **Non-positivity**, the net amount of sand should not increase after a single topple.

$$\forall \ \mathbf{x} \in L \ \ \sum_{\mathbf{y} \in L} \Delta(\mathbf{x}, \mathbf{y}) \leq 0. \tag{A.3}$$

4. **Strictly decreasing**, the net amount of sand should decrease after an avalanche of topples.

$$\sum_{\mathbf{x} \in L} \sum_{\mathbf{y} \in L} \Delta(\mathbf{x}, \mathbf{y}) < 0 \tag{A.4}$$

The **toppling matrix** in the BTW model (for d-dimensions) is defined to be the function.

$$\Delta(x, y) = \begin{cases} -2d & \text{if } x = y \\ 1 & \text{if } x \text{ and } y \text{ are nearest neighbours} \\ 0 & \text{otherwise} \end{cases} \tag{A.5}$$

Indeed we see the above matrix decreases the sand at the toppling site and distributes 1 grain to each nearest neighbour. All other sites remain unaffected.

Now we prove that this toppling matrix satisfies each of the desired properties.

Proposition A.1.1. *The toppling matrix $\Delta(x, y)$ satisfies the symmetry property A.1.*

Proof. Let $x, y \in L$ such that $x \neq y$. then

$$\Delta(x,y) = \begin{cases} 1 & \text{if } x \text{ and } y \text{ are nearest neighbours} \\ 0 & \text{otherwise} \end{cases} = \begin{cases} 1 & \text{if } y \text{ and } x \text{ are nearest neighbours} \\ 0 & \text{otherwise} \end{cases} = \Delta(y,x) \geq 0 \tag{A.6}$$

□

Proposition A.1.2. *The toppling matrix $\Delta(x, y)$ satisfies the self loss property A.2.*

Proof. Let $x \in L$ then

$$\Delta(x,x) = -2d \text{ where we have the dimension } d > 0 \text{ thus } \Delta(x,x) < 0 \tag{A.7}$$

□

Proposition A.1.3. *The toppling matrix $\Delta(x, y)$ for dimension $d = 2$, satisfies the non-positive property property A.3.*[1]

Proof. Let $x \in L$ then the geometry of a 2 dimensional finite lattice implies we have three possible cases for the number of nearest neighbours of x. If x is a corner of the lattice then x has 2 nearest neighbours. Thus

$$\sum_{\mathbf{y} \in L} \Delta(\mathbf{x}, \mathbf{y}) = -2(2) + 1 + 1 = -2 \tag{A.8}$$

Else if x is on an edge of the lattice x but not a corner then x will have 3 nearest neighbours.

$$\sum_{\mathbf{y} \in L} \Delta(\mathbf{x}, \mathbf{y}) = -2(2) + 1 + 1 + 1 = -1 \tag{A.9}$$

Otherwise x is not on an edge of the grid and thus has 4 nearest neighbours.

$$\sum_{\mathbf{y} \in L} \Delta(\mathbf{x}, \mathbf{y}) = -2(2) + 1 + 1 + 1 + 1 = 0 \tag{A.10}$$

Thus for every $x \in L$ we have shown that $\sum_{\mathbf{y} \in L} \Delta(\mathbf{x}, \mathbf{y}) \leq 0$ as desired. □

Proposition A.1.4. *The toppling matrix $\Delta(x, y)$ for dimension $d = 2$, satisfies the strictly decreasing property A.4.*

Proof. Recall from the proof of A.3 that given an $x \in L$, we have

$$\sum_{\mathbf{y} \in L} \Delta(\mathbf{x}, \mathbf{y}) = \begin{cases} -2 & \text{if } x \text{ is a corner of the grid.} \\ -1 & \text{if } x \text{ is on the edge of the grid and not a corner.} \\ 0 & \text{otherwise} \end{cases} \tag{A.11}$$

Now for a rectangular $m \times n$ grid we have that there are 4 corners, $2m + 2n - 4$ edge pieces that are not corners and lastly $m \times n - 2m - 2n$ remaining points. Therefore we have the following inequality:

$$\sum_{\mathbf{x} \in L} \sum_{\mathbf{y} \in L} \Delta(\mathbf{x}, \mathbf{y}) = 4(-2) + (2m + 2n - 4)(-1) + (mn - 2m - 2n)(0) = -4 - 2m - 2n < 0 \tag{A.12}$$

□

The **stability threshold** has value $k = 2d$ for our toppling matrix. This value is defined to be the threshold value that determines when a site is stable or unstable.

[1] Otherwise we would have to consider a general formula for the number of nearest neighbours

A.2 Toppling and Stabilization Operators

Now given a toppling matrix we can define a **toppling operator** T_x which maps a lattice configuration p to p' :

$$(T_x p)(y) = \begin{cases} p(y) + \Delta(x,y), & \text{for } p(x) \geq \Delta(x,x) \\ p(y), & \text{otherwise} \end{cases} \tag{A.13}$$

Proposition A.2.1. *Let p be an unstable configuration, then the two toppling operators T_x and $T_{x'}$ commute.*

Proof. Let $x, x' \in L$ and p be an unstable configuration of L. Then we have the following two toppling operators T_x and $T_{x'}$ defined as in A.13 which change the value of a site $y \in L$ given the toppling of the site x or x' respectively. Checking commutativity of these operators is then a straightforward computation:

$$(T_{x'} T_x)(y) = T_{x'} \left(\begin{cases} p(y) + \Delta(x,y), & \text{for } p(x) \geq \Delta(x,x) \\ p(y), & \text{otherwise} \end{cases} \right) \tag{A.14}$$

$$= \begin{cases} (p(y) + \Delta(x,y)) + \Delta(x',y), & \text{for } p(x) \geq \Delta(x,x) \text{ and } p(x') \geq \Delta(x',x') \\ p(y), & \text{otherwise} \end{cases} \tag{A.15}$$

$$= \begin{cases} (p(y) + \Delta(x',y)) + \Delta(x,y), & \text{for } p(x') \geq \Delta(x',x') \text{ and } p(x) \geq \Delta(x,x) \\ p(y), & \text{otherwise} \end{cases} \tag{A.16}$$

$$= T_x \left(\begin{cases} p(y) + \Delta(x',y), & \text{for } p(x') \geq \Delta(x',x') \\ p(y), & \text{otherwise} \end{cases} \right) \tag{A.17}$$

$$= (T_x T_{x'})(y) \tag{A.18}$$

□

In general multiple topplings are required to stabilize a configuration p. We can introduce a stabilization operator that maps an unstable configuration to a stable configuration

$$\mathbb{T} : U_L \to S_L \tag{A.19}$$

where U_L is the space of all height configurations and $S_L \subset U_L$ is the space of all equal stable height configurations. Such an operator has the explicit form given in terms of toppling operators T_{x_i}:

$$\mathbb{T} = \prod_{i=1}^{N} T_{x_i} \tag{A.20}$$

where N is the number of unstable sites through an avalanche.

Proposition A.2.2. *The stabilization operator $\mathbb{T}$ is well defined.*

Proof. show that the mapping T is well-defined means if we have have two sequences of topplings that stabilize an unstable configuration, then their respective stabilization is unique and the sequence is unique up to permutation of toppling operators. Let $p \in U_L$ be an unstable configuration and let $(x_1, ...x_j)$ and $(y_1, ..y_m)$ be two sets of vertices such that

$$\left(\prod_{i=1}^{j} T_{x_i} \right) u = v, \quad \text{and} \quad \left(\prod_{i=1}^{m} T_{y_i} \right) u = w \tag{A.21}$$

where $v, w \in S_L$. We proceed by induction on $j = 1$. **The base case**: For $j = 1$, the configuration u has one unstable vertex x_1 that when toppled results in a stable configuration. As a result, there exists one sequence of topplings T_{x_1} that results in a unique stable configuration. This proves the base case. **The inductive step** : Assume for all configurations that require k topplings there exists a unique sequence of topplings up to permutation that result in a unique stable configuration. Let $j = k+1$ and $x = (x_1, ..., x_j)$ be the sequence of vertices that need to be toppled for u to be stabilized. Assume, the unstable configuration u has another sequence of vertex topplings $y = (y_1, ..., y_m)$. We need to show that $m = k+1$ and y is a permutation of x. The vertex x_1 is unstable and the

first to be toppled in x so it must also be toppled in the sequence y; let $k \in \{1, ...m\}$ be the smallest such that $x_1 = y_k$. Since vertex x_1 is unstable for u and toppling is commutative, we have

$$\left(\prod_{i=k+1}^{m} T_{y_i}\right) T_{x_1} \left(\prod_{i=k}^{k} T_{y_i}\right) u = \left(\prod_{i=1, i\neq k}^{m} T_{y_i}\right) (T_{x_1})\, u \tag{A.22}$$

Let $T_{x_1} u = v$ then v is an unstable configuration that needs k vertices to be toppled. From our assumption, every configuration needing k vertices to be toppled has a unique stabilization and unique sequence of vertex toppling up to permutation. Thus, the indices $j-1$ and $m-1$ must be equal for v, and the sequences $(x_2, ..., x_j)$ and $(y_1, ..., y_{k-1}, y_{k+1}, ..., y_m)$ are unique up to permutation. □

A.3 A Derivation of Power Laws from Scale Invariance

Definition A.3.1. *Suppose we have a system which is scale invariant with respect to the real variable x. A function $f(x)$ is said to be **scale invariant** with respect to x if when multiplying the argument by a constant scaling factor λ the shape of the function is retained but with a different scale. Mathematically that corresponds to having the following property*

$$f(\lambda x) = \lambda_0 f(x) \;\; \forall \lambda_0, \lambda \in \mathbb{R}_{\geq 0} \text{ and } \forall x \in \mathbb{R} \tag{A.23}$$

Proposition A.3.1. *If our observable is scale invariant we show that it necessarily has the form of a power law.*

Proof.

By the definition of scale invariant

$$f(\lambda x) = \lambda_0 f(x) \;\; \forall \lambda_0, \lambda \in \mathbb{R}_{\geq 0} \text{ and } \forall x \in \mathbb{R} \tag{A.24}$$

To determine all such functions that satisfy the above property differentiate both sides with respect to x

$$\lambda f'(x) = \lambda_0 f'(x) \tag{A.25}$$

Now divide this equation by the original equation and obtain

$$\frac{\lambda x f'(\lambda x)}{\lambda x} = \frac{f'(x)}{f(x)} x \tag{A.26}$$

For this equation to be true for all $\lambda > 0$ we must have that right hand side is equal to a constant. Therefore we obtain the following differential equation

$$\frac{f'(x)}{f(x)} = b \in \mathbb{R} \tag{A.27}$$

integrating both sides yields

$$\log(f(x)) = \log(x) \cdot b + C \text{ where } C \in \mathbb{R} \tag{A.28}$$

Therefore we deduce that our function f must be in the form of a power law

$$f(x) = a x^b \text{ with } a, b \in \mathbb{R} \tag{A.29}$$

Moreover, we can now substitute this f back into our original expression and deduce the explicit dependence of λ_0 on λ

$$a(\lambda x)^b = \lambda_0 a x^b \implies \lambda_0 = \lambda^b \tag{A.30}$$

□

As the function above is arbitrary we have no restriction on the exponent b however if f was a probability density function then the normalization and integrability constraints would impose the exponent be negative.

Bibliography

[1] Per Bak. Catastrophes and self-organized criticality. *American Institute of Physics*, Computers in Physics 5:430–433, 1991.

[2] Per Bak, Chao Tang, and Kurt Wiesenfeld. Bak, p., tang, c. wiesenfeld, k. self-organized criticality: An explanation of 1/f noise. phys. rev. lett. 59, 381-384. *Physical Review Letters*, 59:381–384, 08 1987.

[3] B. Gutenberg and C. F. Richter. Earthquake magnitude, intensity, energy, and acceleration: (Second paper). *Bulletin of the Seismological Society of America*, 46(2):105–145, 04 1956.

[4] Stefan Hegarton. *Self-Organized Criticality in Earth Systems*. Springer-Verlag New York Inc., 2002.

[5] Donald L. Turcotte, Bruce D. Malamud, Fausto Guzzetti, and Paola Reichenbach. Self-organization, the cascade model, and natural hazards. *Proceedings of the National Academy of Sciences*, 99(suppl_1):2530–2537, 2002.