SANDPILES, LANDSLIDES AND SELF-ORGANISED CRITICALITY

SUPERVISOR: JONATHAN FLYNN

AUTHOR: CHENYUAN QU

STUDENT ID: 29975832

BSc report for PHYS3018
Bachelor of Science Project

Physics and Astronomy
University of Southampton
United Kingdom
25/12/2020

SANDPILES, LANDSLIDES AND SELF-ORGANISED CRITICALITY

CHENYUAN QU

Abstract

This article did deep research on property of sandpiles model, *Bak-Tang-Wiesenfeld model*, in one, two and three dimensions. The property of sandpiles in different dimensions was explored. In one dimension, a minimal stable state would be analyzed. Its slope and function of critical state can also be calculated. We also look at the relationship between the grains reaching the stable state and size of grid. In higher dimension, the self-organised stable would be explored instead. Besides the site occupation number, avalanche size distribution was researched which followed the power-law $p(s) \sim cs^{-a}$. Python is used to simulate the models in one and two dimensions and their properties are explained by formula and words. The physical situation based on sandpiles model was also discussed.

1 INTRODUCTION

Granular material is one of most significant subjects in Physics, which is very common in the daily life, like stones, sands, cereals and products storage. In industrial, constructive, agricultural field, there is huge amount of granular material to be produce. Meanwhile, many natural phenomenon can also be predicted by exploiting the concept of granular material. Overall, granular material is a widely broad field. Physicists started their research on it from nineteenth century. Granular material would show some unique property in some condition. To research it more conveniently, many models, such as *Manna model*[1], *Rice pile model*[2] and *Dhar model*[3], have been propounded. *Bak-Tang-Wiesenfeld model* is used to find out its properties in this experiment.

2 THEORETICAL BACKGROUND

Granular material is the set of huge amount of solid grains with more than one micrometer diameter. Granular material has some properties[4]:

i Temperature does not affect the system macroscopically.

ii Fission has become the main force between grains.

iii The most of collision in the system is inelastic, which would lose energy after collisions.

Sandpile is a typical situation of granular material which is defined by, on a flat platform, dropping the grains of sand one by one from above the platform. In the beginning, the grain of sand would remain its position, however, after adding more grains of sand, the sandpile would become increasingly steeper. Then, adding one more grains of sand may cause a part of sand grains to avalanche when the critical height is above the critical value. Meanwhile, the sand grains avalanching would make more grain avalanche. After adding enough number of grains of sand, sandpile would reach a minimal or self-organised critical state. This model was proposed by Bak, Tang and Wiesenfeld and called *Bak-Tang-Wiesenfeld model*.[5]
In *Bak-Tang-Wiesenfeld model*, the system would evolve to the self-organization without adjusting the external variables, and any tiny disturbance would

cause the chain reaction, which may output the energy and its disturbance of avalanche shows the power-law relationship.[6] Self-organised phenomenon is widely in nature and society, like avalanche, earthquake, temperature flow and traffic rules[7][8][9], whose most significant property is that the system would evolve to the critical state.
For example, there is a significant formula called omori formula in physics of earth.[10] This formula illustrates that the probability that aftershock occurs obeys the power-law with the decay of time. The model of earthquake can be explained as that the rock layers of the Earth can be regarded as large amount of plates with different layers, and there is some interaction force between different plates while the energy flows from the Mantle into these plates. The energy will cause a tiny disturbance of one plate and the disturbance may cause the sliding of a small range of plates in the beginning which will also result in the sliding of a larger range of plates and finally it will evolve to a self-organization state. Therefore, our model should be able to reproduce these phenomenon and get some rules for that.
In this article, we will explore the property of sandpiles in different dimensions.

3 ONE-DIMENSIONAL MODEL

In this experiment, we used python to simulate one-dimensional *Bak-Tang-Wiesenfeld model*, which is a very simple model from the reality.

3.1 EXPERIMENTAL METHOD

Imagine in a discrete one-dimension space, there is a flat platform and the grains of sand would drop one by one from above the platform. We can separate the platform to N sites, which is labeled from 0 to $N-1$. With the amount of sand dropping down, some of grains would accumulate on some sites. We can label the height of column from H_0 to H_{N-1}. There is also a critical height z be defined, which shows the height difference between two adjacent sites:

$$z_n = h_n - h_{n-1},$$
$$z_{N-1} = h_{N-1}.$$

When the magnitude of critical height is equal or above the critical value, the avalanche would occur. Here we set the critical value at 3.

When critical height is equal to or higher than 3, the sand would avalanche to left:

$$h_n = h_n - 1,$$

$$h_{n-1} = h_{n-1} + 1.$$

Meanwhile, when critical height is equal or lower than -3, the sand would avalanche to right:

$$h_{n-1} = h_{n-1} - 1,$$

$$h_n = h_n + 1.$$

There are several different kinds we can discover.

3.1.1 MODEL WITH ONE END OPEN AND DROPPING GRAINS FIXEDLY

In this model, there is a board placed on the left end of platform to avoid the grains from leaving the platform from the left side. In the other words, at site 0, there is a closed boundary, at same time, the grains of sand drops from this place above the platform. The rule of avalanche is totally same as above. The grid would be like Figure 3.1 after enough grains have dropped.

The grid shows a slope starting at the left end and decreasing to lowest at the right end. In this period, we can find that, at some time, the shape of grid has already been exactly same as the shape in the end. After that time, wherever the grain drops from, it always rolls down to the right end and leaves the grid. At that time, grid reached its minimally stable state. Repeat the process above at different sizes of grids, record the number of grains dropped to reach the minimally stable state respectively.

3.1.2 MODEL WITH ONE END OPEN AND DROPPING GRAINS RANDOMLY

This model has the same rules as model 3.1.1 except the place where the grains drop. The grains drops from a random place above the platform instead of only the left end. The final grid after 200000 grains dropped is exactly same as 3.1. The numbers of grains dropped to reach the minimally stable state for different sizes of grids are recorded as well.

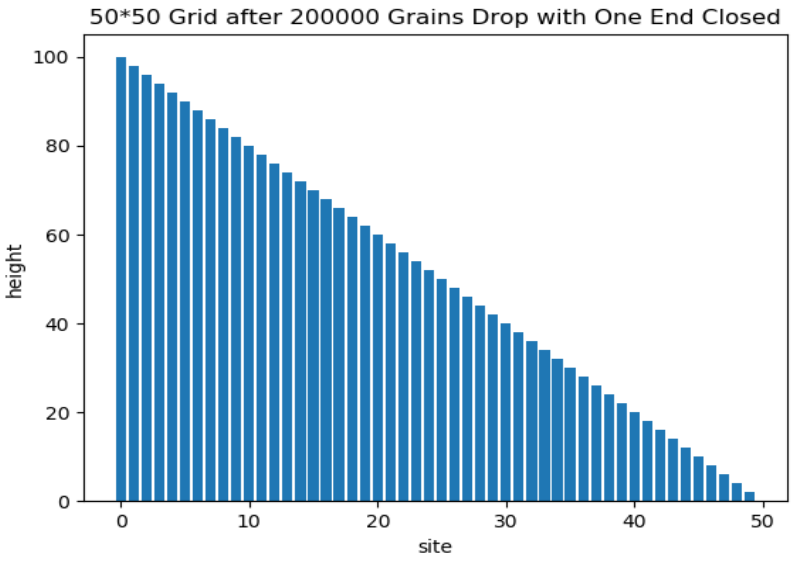

Figure 3.1: 50*50 grid after 200000 grains drop.

3.1.3 MODEL WITH TWO END OPEN AND DROPPING GRAINS FIXEDLY

The difference between this model and model 3.1.1 is that the board placed on the left end would be removed. The rule of avalanche mentioned above is slightly different. The critical height, z, need to detect whether the left end would be avalanche:

$$z_{-1} = h_0.$$

If z_{-1} is equal or above 3, the site 0 would avalanche to left,

$$h_0 = h_0 - 1.$$

The grid would be like Figure 3.2 (Right) after enough grains dropped.

The grid shows that the highest sites are in the middle (site 24) and the height becomes lower while site is closer to these two ends. At the same time, the grid would also reach its minimally stable state in the period. Same as 3.1.1, record the number of grains dropped to reach the minimally stable state for each size of grid.

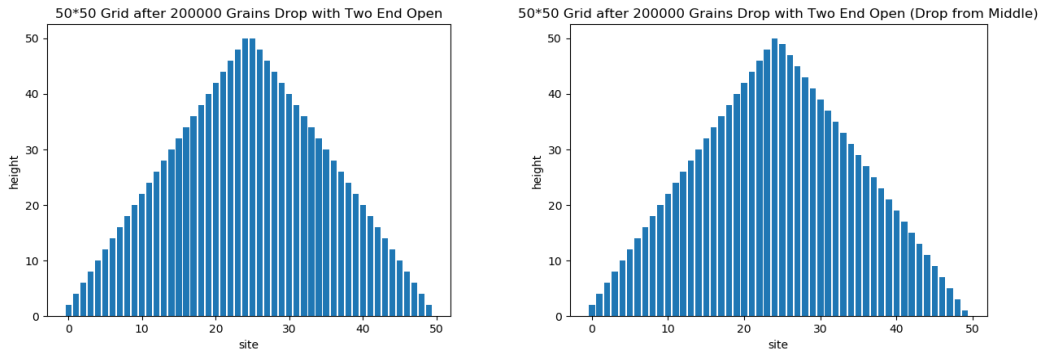

Figure 3.2: 50*50 Grid after 200000 Grains Drop With Two End Open Fixedly (Right) and Randomly (Left).

3.1.4 MODEL WITH TWO END OPEN AND DROPPING GRAINS RANDOMLY

This model is same as model 3.1.3 but the grain of sand will only drop from the middle above the platform. The grid would be like Figure 3.2 (Left). The grid shows very similar to the grid of 3.1.3, but it has two highest sites instead of one. We record the number of grains dropped at critical state for different sizes again.

3.2 THEORETICAL METHOD

To easily observe these models and record the data, Python is used to simulate these models. The grid is represented by a list in the code:

```
grid = np.zeros(grid_size_number, int)
```

Dropping the grain is equivalent to adding one on the grid list randomly or fixedly. After that, check which sites are about to avalanche by traversing the grid list and avalanche on each site. We then check every site again to ensure no site is about to avalanche. If there is still some sites about to avalanche, run the processes above again until no site about to avalanche. The whole process is regarded as an complete avalanche. I program a recursion function here:

```
def avalanche(grid, coordinates_left, coordinates_right,
              grid_size_number, index_of_site):
```

```
    grid, index_of_site = avalanche_separate(grid, coordinates_left,
                                             coordinates_right,
                                             grid_size_number,
                                             index_of_site)

    coordinates_left, coordinates_right = find_coordinates(grid)

    if not (coordinates_left == [] and coordinates_right == []):
        grid, index_of_site = avalanche(grid, coordinates_left,
                                        coordinates_right,
                                        grid_size_number,
                                        index_of_site)

    return grid, index_of_site
```

After an entire avalanche is completed, drop another grain and do an avalanche check again. In the program, I also add a detector to check whether the grid before the grain dropped and the grid after that are same. If so, compare it with the grid after next grain dropped until the final grid is produced. Therefore, we can know when the grid has reached its minimally stable state and record this for the further research.

3.3 DISCUSSION OF RESULTS

3.3.1 MODEL WITH ONE END OPEN AND DROPPING GRAINS FIXEDLY

We get the number of grains of sand to its minimally stable state for different sizes of grids, whose length are 10, 25, 50, 100, 200 respectively. Therefore, we can plot the number of grains against the length of grid (Figure 3.3)

The graph shows that N and L have exponential relationship, so a log-log graph is plotted as Figure 3.4. Meanwhile, a fitting line is also drawn to show the equation of the relationship between them.

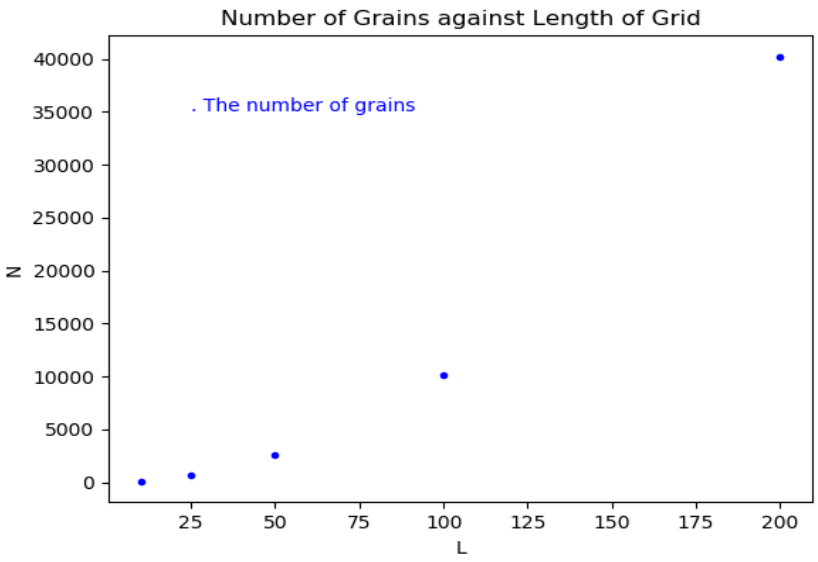

Figure 3.3: Plot graph between number of grains to reach its minimally stable state for model 3.1.1, N, and the length of grid, L.

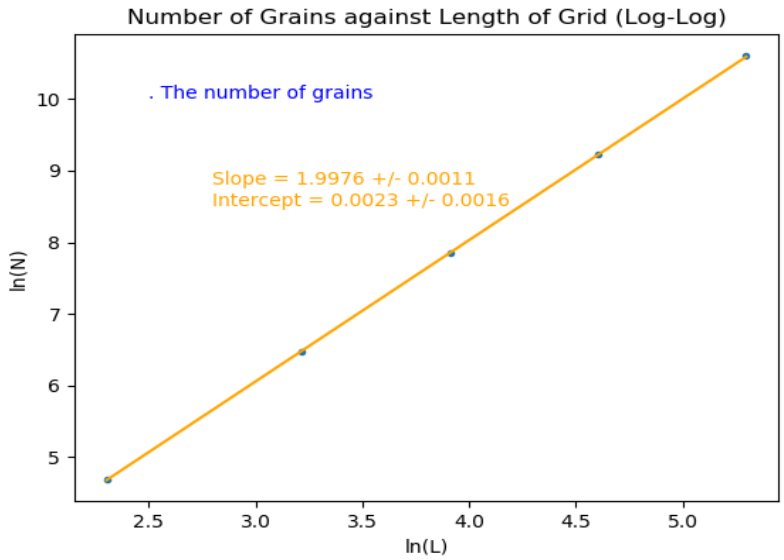

Figure 3.4: Log-Log plot between N and L for model 3.1.1.

The nature log of N and L shows linearly relationship. The slope of the fitting line is 1.9976 ± 0.0011, and the intercept is 0.0423 ± 0.0016 (both rounded to four significant figures). The slope of line is close to 2, and the intercept is close to 0 while the slope has 2.2σ error and the error of intercept is 1.4σ. We can prove the equation by starting with the volume of the pile, V,

$$V = (z_c - 1)x(L - x) + \frac{1}{2}(z_c - 1)x^2 = \frac{z_c - 1}{2}x(2L - x),$$

where z_c is the critical height between adjacent sites, which is 3 in our model and x is the number of sites in critical section. Then one new grain drops, the volume increases, so:

$$\frac{dV}{dn} = 1 - \frac{x}{L},$$

and differential volume with x:

$$\frac{dV}{dx} = (z_c - 1)(L - x).$$

Meanwhile, $\frac{dV}{dn} = \frac{dV}{dx}\frac{dx}{dn}$, therefore,

$$\frac{dn}{dx} = (z_c - 1)L.$$

We can calculate the number of grains by integrating it,

$$N = \int_0^L (z_c - 1)L dx = (z_c - 1)L^2. \tag{3.1}$$

Therefore, we can see the relationship between N and L. The slope is $(z_c - 1)$, which is $3 - 1 = 2$, which fits the slope of the fitting line, and the intercept is 0, which fits either.

3.3.2 MODEL WITH ONE END OPEN AND DROPPING GRAINS RANDOMLY

Again plot the log-log graph between N and L for this model (Figure 3.5). We can see that the slope is 1.9808 ± 0.0043, which is close to 2 with 4.47σ. It is same as the slope of last model, but different with the intercept. Therefore, we can deduce the equation between N and L is like,

$$N = (z_c - 1)L^2 + A. \tag{3.2}$$

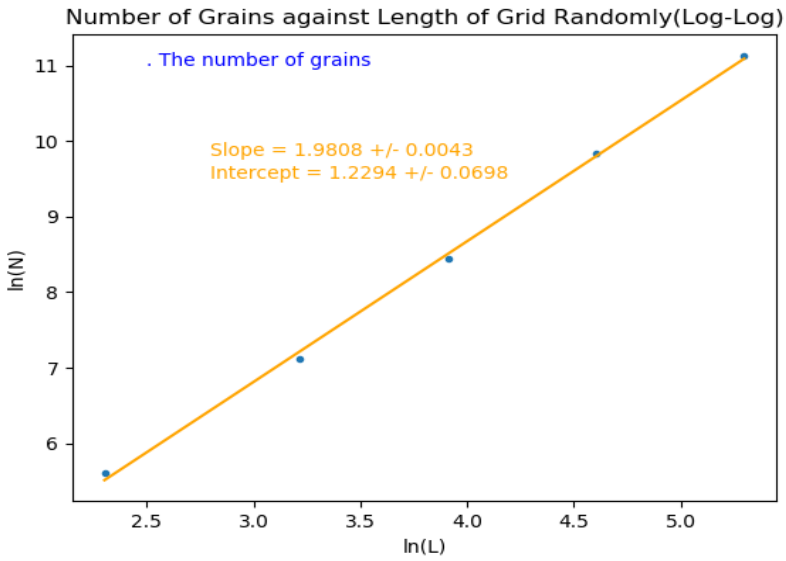

Figure 3.5: Log-Log plot between N and L for model 3.1.2 (drop randomly).

3.3.3 MODEL WITH TWO END OPEN AND DROPPING GRAINS FIXEDLY

After plotting the log-log graph for number of grains to reach its minimally stable and length of grid (Figure 3.6), we can see the same pattern for the fitting line equation as 3.3.1. The slope of line is 1.9978 ± 0.0030, consistent with 2, and intercept is -0.0404 ± 0.0092, which is 4.4σ from 2.
Therefore, we can draw a conclusion that when we drop the grains from above the highest sites in the minimally stable state, the relationship between number of grains and length of grid meets equation (3.1).

3.3.4 MODEL WITH TWO END OPEN AND DROPPING GRAINS RANDOMLY

Firstly, as we look back to Figure 3.2. We will notice that while the length of grid is odd, model 3.2 and model 3.3 have same minimally stable state for same size of grid, however, while the length is even, the former stable state is still symmetrical but the last one is not. This phenomenon shown

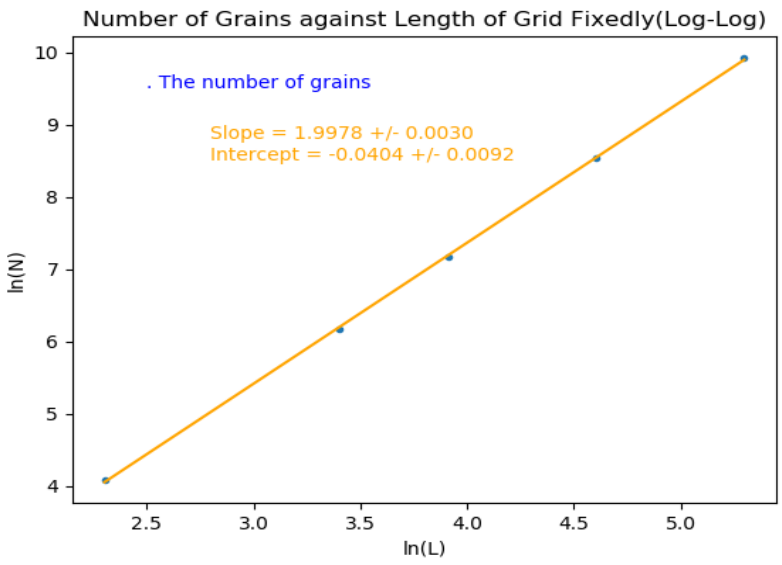

Figure 3.6: Log-Log plot between N and L for model 3.1.3 (drop fixedly).

is because when length is odd, the model rules have a reflection symmetry about the midpoint, and the position of dropping grain is symmetrical to the grid, therefore its final stable state is symmetrical as well. On the contrary, when length is even, the position of dropping grain cannot be exactly in the middle (position must be above one site), so its minimally stable state will not be symmetrical either.

Then we realize that the relationship between this model and last model. It is exactly same as that between model 3.1.1 and model 3.1.2. Plot a log-log graph between N and L again. (Figure 3.7 The formula of fitting line has slope which is 1.9917 ± 0.0031 and close to 2 with 2.68σ. Equation (3.2) can be deduced by this again.

Overall, comparing with model 3.1.1/3.1.2 and model 3.1.3/3.1.4, we can prove that the slope of the fitting line always fits $z_c - 1$ whether the grains drop randomly or fixedly. However, due to the random drop, the fitting lines of model 3.1.2 and 3.1.4 have one constant need be added.

Meanwhile, we can find whatever model we used for one-dimension model, only the minimally stable state will showed instead of self-organised stable state, which is because every grain leaves the grid after the grid reaching the minimally stable state. I will explain it further in the following chapter.

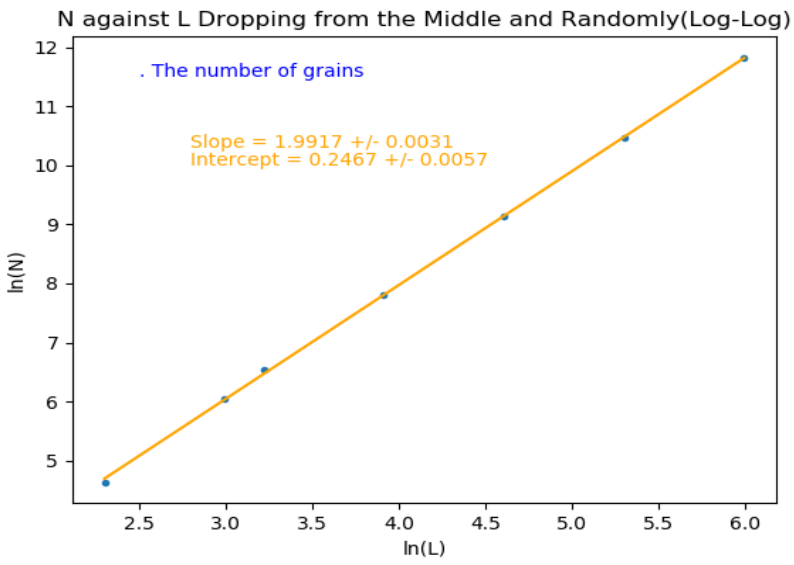

Figure 3.7: Log-Log plot between N and L for model 3.1.4 (dropping from the middle and randomly).

4 TWO-DIMENSIONAL MODEL

For two-dimensional BTW model, Python is still used to simulate.

4.1 EXPERIMENTAL METHOD

Consider a two-dimension square platform.[11] We can divide the platform into N*N sites, where N is the side length of the platform. We label each site from $N_{0,0}$ to $N_{N-1,N-1}$. For each square, it can contain 0, 1, 2, 3, ... grains but less than critical height, h_c, which is set to be 4 in our experiment. The height of them is labeled as $h_{0,0}$ to $h_{N-1,N-1}$. Again, we drop the grains above one site randomly or fixedly. If all sites contain less than h_c grains, the next grain drops. Conversely, if there are some sites containing more than or equal to h_c grains, one avalanche starts:

1. There are four grains leaving the toppling site and going to the four

adjacent sites. If the site which is about to avalanche is on the side or the edge, one or two grains will leave the grid because there is no boundary on the sides of the grid,

$$h_{n,m} = h_{n,m} - 4,$$

$$h_{n-1,m} = h_{n-1,m} + 1,$$
$$h_{n+1,m} = h_{n+1,m} + 1,$$
$$h_{n,m-1} = h_{n,m-1} + 1,$$
$$h_{n,m+1} = h_{n,m+1} + 1,$$

where n and m are greater than 1.

2. After step 1, check all the sites again. If there are still some sites containing above 3 grains, repeat step 1. Until all sites contain less than 4 grains, this avalanche is regarded as completed. Then drop another grain and check grid again.[3] [12]

Meanwhile, we also need to record the mean density and size of avalanche in the experiment. Mean density means the mean density for all the sites. Mathematically,

$$mean\ density = \frac{total\ grains\ remain\ in\ the\ grid}{number\ of\ sites}.$$

The size of avalanche is the number of sites toppling in this whole avalanche process.

Here we design two models with some difference to be researched.

4.1.1 MODEL DROPPING THE GRAINS RANDOMLY

In this model, the grains drop above a random site. After a number of grains dropped, the grid is shown like Figure 4.1 (left). Meanwhile, mean density is also recorded after every grains dropped, so we can plot a graph to show that the mean density changes with time (we can consider one grain dropped as one time unit) as Figure 4.1 (right). Apart from that, we still record the sizes of every avalanche for the further research and repeat the processes above in different sizes of grids.

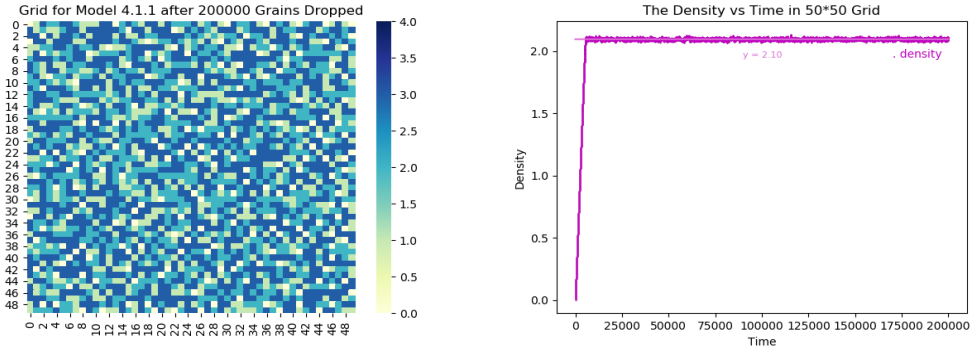

Figure 4.1: Grid for Model 4.1.1 after 200000 Grains Dropped (Left), mean density of it changes with time (Right).

a rough or fragmented geometric shape that can be split into parts, each of which is (at least approximately) a reduced-size copy of the whole

4.1.2 MODEL DROPPING THE GRAINS FIXEDLY

This model obeys the same rule as model 4.1.1 but drop the grains above the middle site of the grid. As the processes done before, the grid after 2000 grains dropped and mean density from beginning to 200000 grains dropped vs time graph are shown as Figure 4.2. We can see that the shape of the grid is split into parts and each of parts is a reduced-size copy of whole, which shows that it is fractal because it is the definition of fractal shape.[13] This property is different with the last model. Sizes of every avalanche are also recorded, and repeats are done for different sizes of grids.

4.2 THEORETICAL METHOD

For two-dimension model, the grid is represented by a matrix:

```
grid = np.zeros((grid_size_number, grid_size_number), int)
```

Then drop the grains randomly or fixedly, check whether there is some toppling sites and record them, which is similar to one-dimension model. When

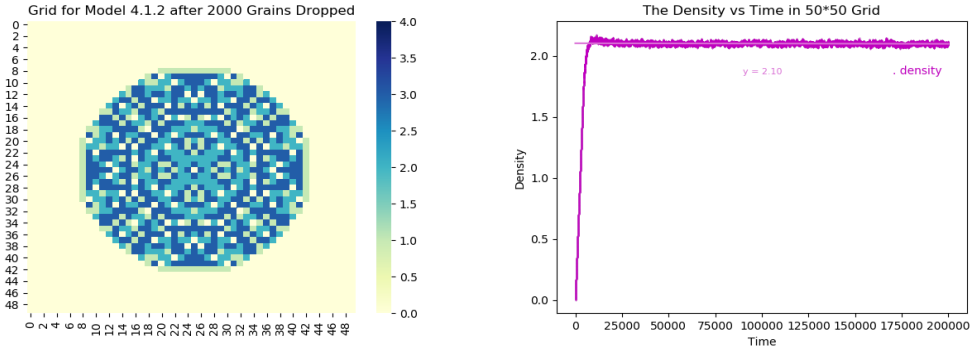

Figure 4.2: Grid for Model 4.1.2 after 2000 Grains Dropped (Left), mean density of the grid changes of it with time (Right).

the avalanche happened, there is one more process to do. The purpose is to detect whether the grain goes out of the grid to avoid error:

```
def verify_if_out_of_matrix(site, number_of_size):
    for coor in site:
        if coor > (number_of_size - 1) or coor < 0:
            return False
    return True
```

To record the mean density and size of avalanche in the whole process, I create two empty lists for them respectively. A variable is also created for size of avalanche and reset in the beginning of every loop. In the end of every loop, a number will be added to each list for recording the situation after this grain dropped. Therefore, after N times repeats, the length of list will be N.
To visualise the grid, I use the heat map to make it beautiful. A save function is also used in order not to repeat the simulation too many times and to compare each data easily.

4.3 DISCUSSION OF RESULT

4.3.1 MINIMALLY AND SELF-ORGANISATION STABLE STATES

When comparing both one-dimension model and two-dimension model, we can clearly see self-organisation stable state is only in two-dimension model.

In other real experiment (rice model[2], for example), the grid always show the self-organisation property, which means that self-organisation should be appeared in one-dimension model, but it is not. That is because that one-dimension model does not have probability of avalanche, which I will explain more in following chapter.

4.3.2 MEAN DENSITY AND STEADY DENSITY

As we have already seen in Figure 4.1 and Figure 4.2, the mean density would remain in a constant while the time becomes greater. This status is also defined as its steady density. For Model 4.1.1, when we look at the graph between mean density and time for different sizes of grids, we would find that 1) the mean density would tend to one constant in all different sizes of grids 2) while size of grid increase, the steady density increases followed. (Left of Figure 4.3) We recorded the data of steady density for different sizes of grids, and put them into one graph. It is observable that the steady densities fit the fitting function well which is an inverse proportional function. Therefore, the steady density increases rapidly when size of grid is small, and approach to one constant, which is 2.11 in this model, when size of grid goes to infinite. For Model 4.1.2, its mean density vs time graph is similar to Figure 4.3 (Left), but the steady density vs size for the grid has some difference. (Figure 4.4) The steady density would approach to a greater constant. This is because the grains drop from the middle, which is farthest point from the edge, the grains will be more unlikely to leave the grid. The coefficient of $\frac{1}{x}$ is more negative which means that the effect of size changing is smaller. This may be caused by the fact that the fractal transformation is not affected by size of grid.

4.3.3 AVALANCHE SIZE DISTRIBUTION

The distribution of size of avalanche is defined as the probability density for avalanches of that size, $D(s)$. The relationship between the size and its distribution is

$$D(s) \simeq s^{-\gamma}.$$

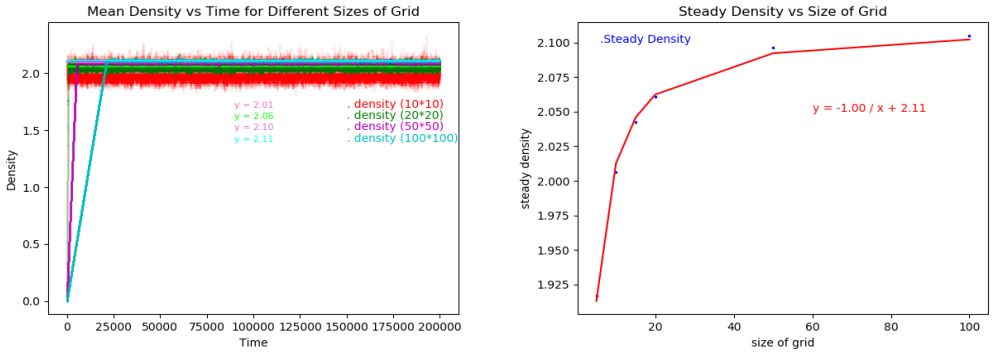

Figure 4.3: Mean density changes with time for different sizes of grids for Model 4.1.1 (Left), the graph between steady density and size of grid (Right).

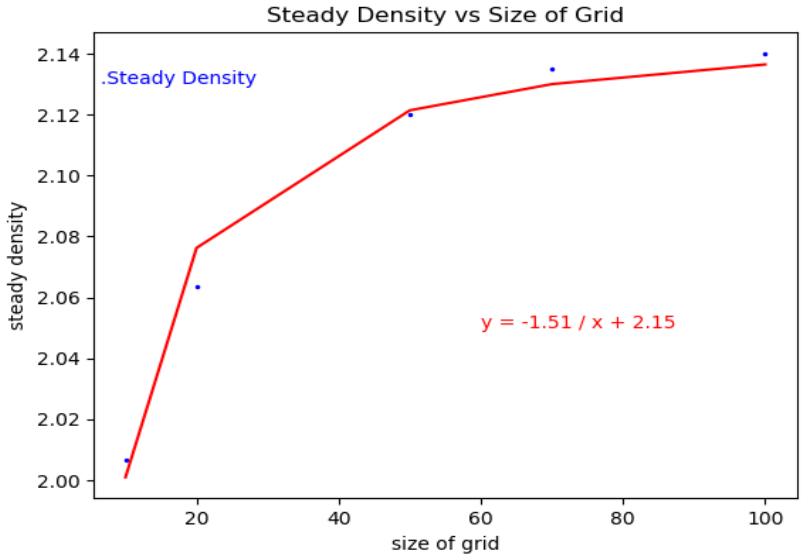

Figure 4.4: The graph between steady density and size of grid for Model 4.1.2.

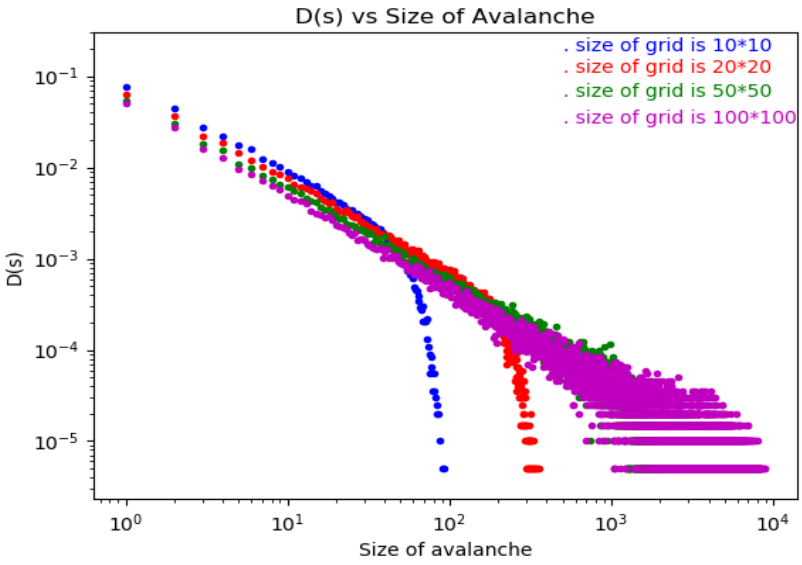

Figure 4.5: The log-log graph between s and D(s) for several sizes of grids, where s stands for size of avalanche, D(s) is the probability of that size appearing.

The distribution of the avalanche size is plotted against the size of avalanche as Figure 4.5 for example. The dots located represent the data, which is very noise especially when the grid becomes bigger. To read the data clearly and fit them easily, we need to simply these data by averaging ranges of s values. *Logarithmic binning* is used based on the anticipation that the range will be getting bigger while the size is huge.[15]

4.3.4 LOGARITHMIC BINNING

In logarithmic binning, we need to divide $logs$ to N bins, $s_0, s_1, ..., s_{N-1}$ and

$$logs_{i+1} - logs_i = w,$$

where w is positive because $s_{i+1} > s \geq s_i$.
Because the sizes of avalanches are integers, so for ith bin, it has

$$n_{i,max} = floor(s_{i+1})$$

N	γ	γ_{error}
10	1.005	0.000
20	1.032	0.001
30	1.042	0.001
50	1.060	0.002
70	1.065	0.005
100	1.081	0.005
200	1.103	0.013
500	1.140	0.022

Table 4.1: γ for different sizes of grids

$$n_{i,min} = floor(s_i) + 1.$$

Therefore, when p(n) is not 0, the average value in that bin is

$$\langle logp \rangle_i = \frac{\sum_{n_{i,min}}^{n_{i,max}} logp(n)}{n_{i,max} - n_{i,min} + 1}. \tag{4.1}$$

The centre of ith bin in the logarithmic axes is

$$\frac{logs_i + logs_{i+1}}{2} = logs_i + \frac{w}{2}.$$

4.3.5 POWER-LAW SCALING

As we used logarithmic binning for the graph, the data is as Figure 4.6. Now, we can use the power-law to fit them,[6][14]

$$D(s) \simeq s^{-\gamma}.$$

We can see the γ for some sizes of grids in the graph. The full σ data is on Table 4.1.

From the table, it is easy to find that *gamma* increases while the N increases but they are not proportional. We can also find the relationship between N and γ by plotting a graph between them. When they are plotted in the log-log graph between $N^{-1/4}$ and γ, the dots are located nearly on a straight line. (Figure 4.7) We set $N^{-1/4}$ as x-axis because we want to figure out the value of γ when N goes to infinity. We fit the data and get the fitting line. The equation for it is

$$\gamma = 1.130(\pm 0.0015) * \exp\left(-0.125(\pm 0.0016) * N^{-1/4}\right).$$

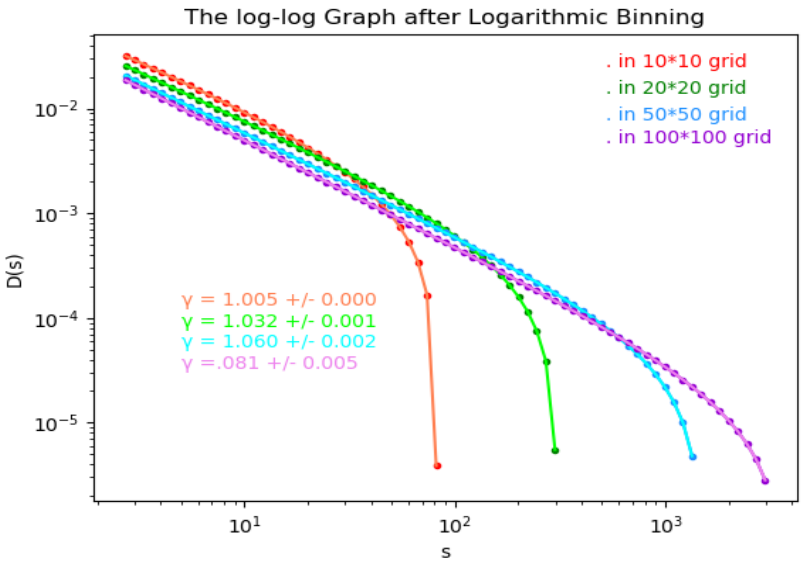

Figure 4.6: The log-log graph between s and D(s) after the logarithmic binning, the dots are data and the lines are fitting lines.

From the equation, we can deduce when N goes to infinite, $N^{-1/4}$ goes to zero, so γ goes to 1.130 ± 0.0015 as followed.

4.3.6 FRACTAL RULE

The model 4.1.2 is clearly fractal. Base on fractal rule,[13]

$$y = ax^k.$$

This rule is also a kind of power-law. Therefore, we can also use logarithmic binning for the data and plot the log-log graph between D(s) and s (similar to Figure 4.6), so we can get the k value which is equivalent to γ. The value would be like Table 4.2, and Figure 4.8 shows the graph between $N^{-1/4}$ and k.

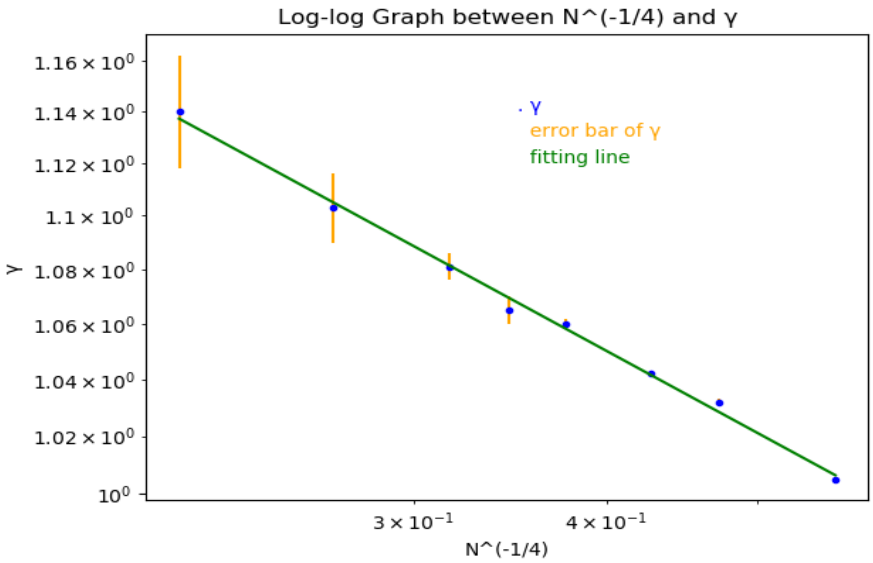

Figure 4.7: The log-log graph between N and γ, where dots are γ, orange lines are error bar of it and the green line is the fitting line.

N	k	k_{error}
10	1.006	0.000
20	1.030	0.001
30	1.042	0.001
50	1.059	0.001
70	1.070	0.003
100	1.082	0.004
200	1.104	0.007
500	1.137	0.012

Table 4.2: k for different sizes of grids

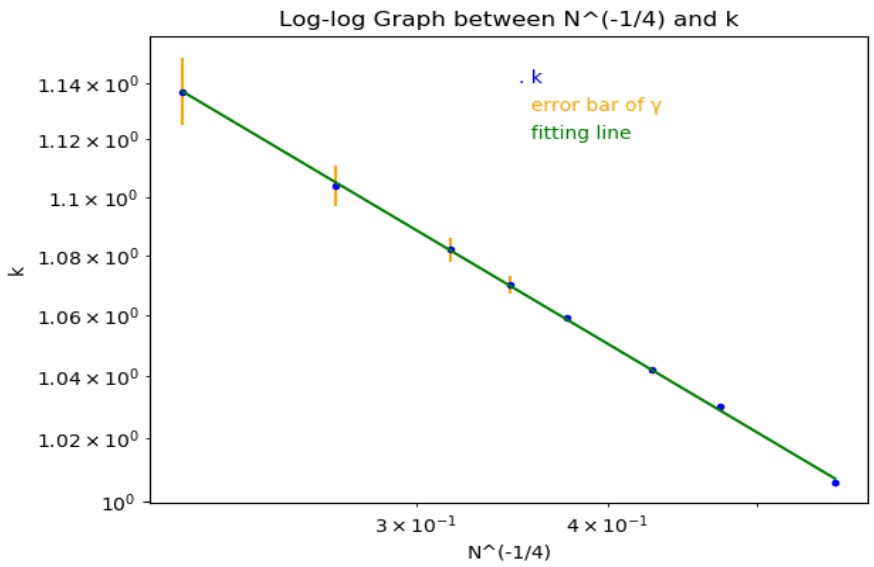

Figure 4.8: The log-log graph between N and k, where dots are k, orange lines are error bar of it and the green line is the fitting line.

The equation for k is

$$k = 1.130(\pm 1.3e-06) * \exp\left(-0.124(\pm 1.3e-06) * N^{-1/4}\right).$$

Compare fractal model with model 4.1.1, k has the similar value and similar relationship with $N^{-1/4}$ with γ, however, it has less error range and the data k is closer to the fitting line. I suppose that for fractal model, the size of avalanche is periodic, so the value of k we can get is more accurate.

4.3.7 SCALING COLLAPSE AND FINITE SIZE SCALING

Looking back to the Figure 4.6, we find that the starting points and the first half points for each grid are nearly parallel but not coincident. Nevertheless, we can set the x axis as $1/L^2$ and a multiplication for $D(s)$ to line up them. Figure 4.9 is plotted by using the equation:

$$D(s)L^{\sigma} \propto \frac{s}{L^{d}}, \tag{4.2}$$

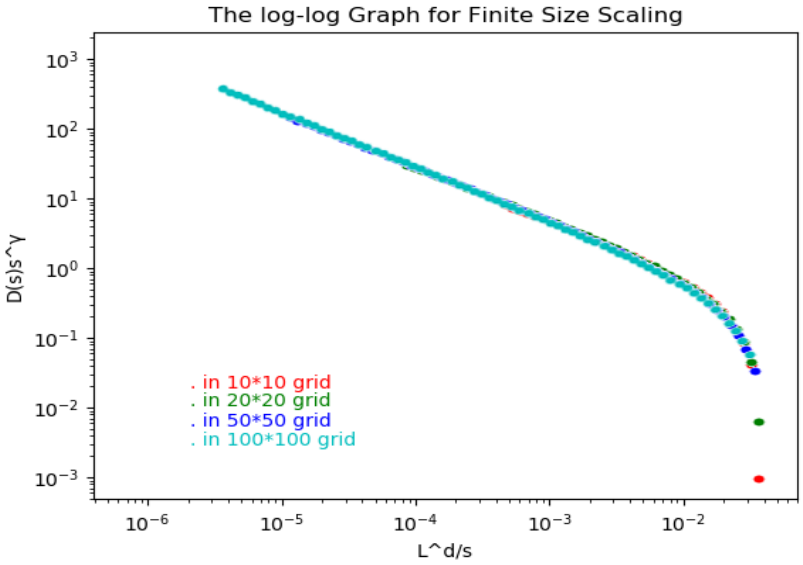

Figure 4.9: The log-log graph for scaling collapse of avalanche area distribution.

where $\sigma = 2.4$ and $d = 2.06$.
This graph shows that all D(s) can be lined up which is unrelated to size of grid. Meanwhile, we also need to research D(s) for infinite large size of grid, which is not possible to be found out through experiments because it need infinite repeat runs. However, *finite size scaling* can be used to achieve it as well as by using data of finite sizes of grids. When we apply finite size scaling, the relationship holds (Figure 4.10)

$$D(s) = s^{-\gamma} f(L^d/s), \tag{4.3}$$

where f is a function, $\gamma = 1.17$, $d = 2.06$ again and L is a characteristic length [11].
Combining equation 4.2 and equation 4.3. The d is same and $\sigma/\gamma = d$, so d should be $d = 2.4/1.17 = 2.05$ which is in the range 2.06 - 2.052. Meanwhile, $\gamma = 1.13$ is only 2.66σ from $\gamma = 1.17$, which proves our result again.

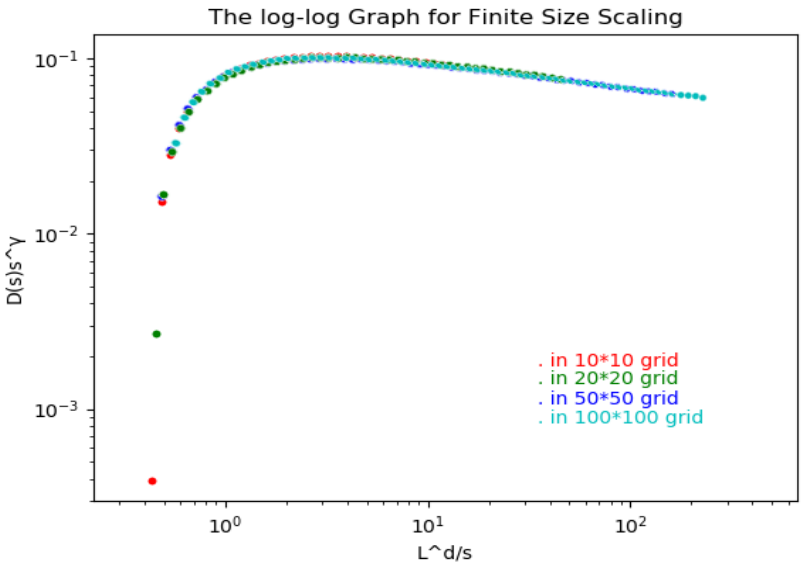

Figure 4.10: The log-log graph for finite size scaling

5 CONCLUSION

1. In one-dimension model, regardless of the position of grains dropping and one or two boundaries opening, the number of grains to reach its minimally stable state will increase followed by size of grid increasing. When the grains drop fixedly, the relationship increases by followed the equation, $N = (z_c - 1)L^2$, where z_c is the set critical height. On the other side, when grains drop randomly, the equation will have a constant, $N = (z_c - 1)L^2 + A$. There are one or two opening boundaries not affecting the relationship if the grains drop from the highest site of the final stable state. Moreover, based on symmetrical rule, if the position of dropping grains is symmetrical, the final stable state would be symmetrical as well and vice versa.

2. In two-dimension model, the mean density would tend to one constant, which is defined as steady density. Steady density increases while the size of grid increases, which is followed by an inverse function. The inverse function also means that when size of grid goes infinite, the size of grid will tend to one constant. For the model with dropping grains

randomly, the constant is 2.11. For the model with dropping grains fixedly, it is 2.15.

3. In two-dimension model, there is a distribution of size of avalanche, which is why two-dimension model has self-organisation property but one-dimension one does not. The relationship between D(s) and size follows the power-law $D(s) \simeq s^{-\gamma}$. After calculating the γ for different sizes of grids, we can also find that when size of grid is increasingly larger, γ goes greater as well, and through researching the relationship between γ and $N^{-1/4}$, we can get the value of γ when size of grid is infinite, which is about 1.130. Moreover, scaling collapse and finite size scaling are used, and the value of γ is also proved again. In Model 4.1.2, the grid shows fractal property while the sand drop. So the fractal rule should be applied on it, $y = ax^k$, which is also a kind of power-law equation. We get a similar k value with γ value and similar relationship with $N^{-1/4}$, but with less error and fitting better. This is because the size of avalanche for fractal model is periodic, so power variable is easier to find out.

References

[1] Gary Willis, Gunnar Pruessner
Spatio-temporal Correlations in the Manna Model in one, three and five dimensions
https://doi.org/10.1142/S0217979218300025

[2] Vidar Frette, Kim Christensen, Anders Malthe-Sørenssen, Jens Feder, Torstein Jøssang & Paul Meakin
Avalanche dynamics in a pile of rice
Nature volume 379, pages49–52(1996).

[3] Dhar, D
Theoretical studies of self-organized criticality
Phys. A 369, 29-70.(2006)

[4] Duran, J.
Sands, Powders, and Grains: An Introduction to the Physics of Granular Materials (translated by A. Reisinger)

November 1999, Springer-Verlag New York, Inc., New York, ISBN 0-387-98656-1.

[5] Bak P; Tang C; Wiesenfeld K.
Physical review letters [Phys Rev Lett], 1987 Jul 27: Self-organized criticality: An explanation of the 1/f noise
American Physical Society, 1987.

[6] N. H. Bingham, C. M. Goldie, and J. L. Teugels
Regular variation
Cambridge University Press, 1989

[7] Ashby, W. R. (1947)
Principles of the Self-Organizing Dynamic System
The Journal of General Psychology. 37 (2): 125–28. doi:10.1080/00221309.1947.9918144

[8] Scott Camazine, Jean-Louis Deneubourg, Nigel R. Franks, James Sneyd, Eric Bonabeau, Guy Theraula
Self-organization in Biological Systems
Princeton University Press, 17/09/2003, Page 538

[9] Glansdorff, P., Prigogine, I. (1971)
Thermodynamic Theory of Structure, Stability and Fluctuations
Wiley-Interscience, London. ISBN 0-471-30280-5

[10] Utsu T, Ogata Y
The centenary of the omori formula for a decay law of aftershock activity
Journal of Physics of the Earth (1995) 43(1) 1-33

[11] Bak, P., Tang, C. and Wiesenfeld, K
Self-organized criticality
Phys. Rev. A 38, 364-374(1988).

[12] Redig, F
Mathematical aspects of Abelian sandpile models
Course 14, 657-729(2006).

[13] Benoit B. Mandelbrot
The Fractal Geometry of Nature
Henry Holt and Company, 1983. Page 468

[14] K Christensen, HC Fogedby, and HJ Jensen
Dynamical and spatial aspects of sandpile cellular automata
J Stat Phys 63, 653–684 (1991) DOI: 10.1007/BF01029204

[15] Stefan Hergarten (2013)
Self-Organized Criticality in Earth Systems
Springer Berlin Heidelberg, Page 36-38

6 APPENDIX

6.1 ONE-DIMENSION MODEL SIMULATION CODE

```
import numpy as np
import time
import copy
import os

import sys

sys.setrecursionlimit(1000000000)


def dropping_sand(grid):
    if (len(grid) % 2) == 0:
        colomn = int(len(grid) / 2 - 1)
    if (len(grid) % 2) == 1:
        colomn = int((len(grid) - 1) / 2)
    grid[colomn] += 1
    return grid


def find_coordinates_left(grid):
    coordinates_left = []
    for x in range(1, grid.shape[0]):
        if grid[x] - grid[x - 1] > 2:
            coordinates_left.append(x)
    if grid[0] > 2:
        coordinates_left.append(0)
    return coordinates_left


def find_coordinates_right(grid):
    coordinates_right = []
    for x in range(grid.shape[0] - 1):
        if grid[x] - grid[x + 1] > 2:
            coordinates_right.append(x)
```

```
    if grid[-1] > 2:
        coordinates_right.append(len(grid) - 1)
    return coordinates_right


def find_coordinates(grid):
    coordinates_left = find_coordinates_left(grid)
    coordinates_right = find_coordinates_right(grid)

    return coordinates_left, coordinates_right


def verify_if_out_of_matrix(point, grid_size_number):
    if point > (grid_size_number - 1) or point < 0:
        return False
    return True


def avalanche_separate_left(grid, singlecoordinate, grid_size_number,
                            index_of_site):
    grid[singlecoordinate] -= 1
    index_of_site.append(singlecoordinate)
    grid_left = singlecoordinate - 1

    if verify_if_out_of_matrix(grid_left, grid_size_number):
        grid[grid_left] += 1
        index_of_site.append(grid_left)

    return grid, index_of_site


def avalanche_separate_right(grid, singlecoordinate, grid_size_number,
                             index_of_site):
    grid[singlecoordinate] -= 1
    index_of_site.append(singlecoordinate)
    grid_right = singlecoordinate + 1

    if verify_if_out_of_matrix(grid_right, grid_size_number):
```

```
            grid[grid_right] += 1
            index_of_site.append(grid_right)

        return grid, index_of_site


    def avalanche_separate(grid, coordinates_left, coordinates_right,
                           grid_size_number, index_of_site):
        for single_coordinate_left in coordinates_left:
            grid, index_of_site = \
                avalanche_separate_left(grid, single_coordinate_left,
                                        grid_size_number, index_of_site)
        for single_coordinate_right in coordinates_right:
            grid, index_of_site = \
                avalanche_separate_right(grid, single_coordinate_right,
                                         grid_size_number, index_of_site)

        return grid, index_of_site


    def avalanche(grid, coordinates_left, coordinates_right,
                  grid_size_number, index_of_site):
        grid, index_of_site = \
            avalanche_separate(grid, coordinates_left, coordinates_right,
                               grid_size_number, index_of_site)

        coordinates_left, coordinates_right = find_coordinates(grid)

        if not (coordinates_left == [] and coordinates_right == []):
            grid, index_of_site = avalanche(grid, coordinates_left,
                                            coordinates_right, grid_size_number,
                                            index_of_site)

        return grid, index_of_site


    def save_data(grid_size_number, repeat_times, critical_number,
                  site_number_total, number_of_grains_total):
```

```
    save_folder = r'D:\Study\Physics\Sandpiles\Data'
    t = time.strftime('%Y_%m_%d_%H%M%S', time.localtime())
    save_path = os.path.join(save_folder,
                             '1D_fixed' + str(grid_size_number) + '_' +
                             str(repeat_times) + '_' + t + '.txt')

    text = [grid_size_number, '\n',
            repeat_times, '\n',
            critical_number, '\n',
            site_number_total, '\n',
            number_of_grains_total]
    with open(save_path, 'w+') as txt:
        for item in text:
            txt.write(str(item))
    print('The data is saved')
    return True


if __name__ == '__main__':
    grid_size_number_list = [10, 30, 50, 100, 200, 400, 800]

    for grid_size_number in grid_size_number_list:
        grid = np.zeros(grid_size_number, int)

        number_of_grains_total = []
        site_number_total = []

        repeat_times = 200000
        critical_state = 0
        critical_number_50_plus = None

        last_grid = copy.copy(grid)
        for i in range(repeat_times):

            grid = dropping_sand(grid)

            index_of_site = []
            coordinates_left, coordinates_right = find_coordinates(grid)
```

```
            grid, index_of_site = avalanche(grid, coordinates_left,
                                            coordinates_right,
                                            grid_size_number,
                                            index_of_site)

            site_number_total.append(len(list(dict.fromkeys(index_of_site))))
            number_of_grains_total.append(np.sum(grid))

            if critical_state != 50:
                if all([grid[a] == last_grid[a]
                        for a in range(grid_size_number)]):
                    critical_number_50_plus = i
                    critical_state += 1
                else:
                    critical_state = 0
            else:
                break
            print(grid)
            last_grid = copy.copy(grid)

        if critical_number_50_plus is not None:
            critical_number = critical_number_50_plus - 50
            print('The grid reached critical state at ', critical_number)

        save_data(grid_size_number, repeat_times, critical_number,
                  site_number_total, number_of_grains_total)
```

6.2 TWO-DIMENSION MODEL SIMULATION CODE

```
import numpy as np
from matplotlib import pyplot as plt
import seaborn
import os
import time

import sys

sys.setrecursionlimit(1000000000)
```

```
fig = plt.figure()

plt.ion()


def dropping_sand(grid):
    colomn = np.random.randint(0, grid.shape[0])
    row = np.random.randint(0, grid.shape[0])
    grid[colomn, row] += 1
    return grid


def find_coordinates(grid):
    coordinates = []
    for x in range(grid.shape[0]):
        for y in range(grid.shape[0]):
            if not grid[x, y] < 4:
                coordinates.append((x, y))

    return coordinates


def verify_if_out_of_matrix(site, number_of_size):
    for coor in site:
        if coor > (number_of_size - 1) or coor < 0:
            return False
    return True


def avalanche_separate(grid, singlecoordinate, size_number, index_of_site):
    grid[singlecoordinate] -= 4
    index_of_site.append(singlecoordinate)
    grid_top = (singlecoordinate[0] - 1, singlecoordinate[1])
    grid_bottom = (singlecoordinate[0] + 1, singlecoordinate[1])
    grid_left = (singlecoordinate[0], singlecoordinate[1] - 1)
    grid_right = (singlecoordinate[0], singlecoordinate[1] + 1)
```

```
    for point in [grid_top, grid_bottom, grid_left, grid_right]:
        if verify_if_out_of_matrix(point, size_number):
            grid[point] += 1

    return grid, index_of_site


def plot_grid(grid):
    ax = seaborn.heatmap(grid, cmap='YlGnBu', vmin=0, vmax=4,
                         xticklabels=2, yticklabels=2, cbar=False)
    plt.plot()
    plt.pause(100)


def avalanche(grid, coordinates, size_number, index_of_site):
    for single_coordinate in coordinates:
        grid, index_of_site = avalanche_separate(grid, single_coordinate,
                                                 size_number, index_of_site)
    coordinates = find_coordinates(grid)
    plot_grid(grid)

    if not coordinates == []:
        grid, index_of_site = avalanche(grid, coordinates, grid_size_number,
                                        index_of_site)

    return grid, index_of_site


def verify_finish(grid):
    if all([number == 3 for list in grid for number in list]):
        return True
    else:
        return False


def save_data(grid_size_number, repeat_times, site_number_total):
    save_folder = r'D:\Study\Physics\sandpiles\data'
    t = time.strftime('%Y_%m_%d_%H%M%S', time.localtime())
```

```
    save_path = os.path.join(save_folder, str(grid_size_number) +
                             '_' + str(repeat_times) + '_' + t + '.txt')

    text = [grid_size_number, '\n',
            repeat_times, '\n',
            site_number_total]
    with open(save_path, 'w+') as txt:
        for item in text:
            txt.write(str(item))
    print('The data is saved')
    return True


def save_density(grid_size_number, repeat_times, density_total):
    save_folder = r'D:\Study\Physics\sandpiles\data'
    t = time.strftime('%Y_%m_%d_%H%M%S', time.localtime())
    save_path = os.path.join(save_folder,
                             'density_' + str(grid_size_number) + '_' +
                             str(repeat_times) + '_' + t + '.txt')

    text = [grid_size_number, '\n',
            repeat_times, '\n',
            density_total]
    with open(save_path, 'w+') as txt:
        for item in text:
            txt.write(str(item))
    print('The data is saved')
    return True


if __name__ == '__main__':

    grid_size_number_list = [50]
    for grid_size_number in grid_size_number_list:

        grid = np.zeros((grid_size_number, grid_size_number), int)
        ax = seaborn.heatmap(grid, cmap='YlGnBu', vmin=0, vmax=4,
                             xticklabels=2, yticklabels=2)
```

```
plt.xlabel('n')
plt.ylabel('m')
plt.title('Heat Map for Two-Dimension Model Simulation')
site_number_total = []
repeat_times = 200000

desity_total = []

for i in range(repeat_times):

    index_of_site = []

    grid = dropping_sand(grid)
    coordinates = find_coordinates(grid)
    grid, index_of_site = avalanche(grid, coordinates,
                                    grid_size_number, index_of_site)
    site_number_total.append(len(list(dict.fromkeys(index_of_site))))

    desity_total.append(np.sum(grid))

    print(i, '\n', grid)

print(grid_size_number, repeat_times, site_number_total)
plot_grid(grid)
save_density(grid_size_number, repeat_times, desity_total)
save_data(grid_size_number, repeat_times, site_number_total)
```