Self-Organised Criticality in Sand

Name: Thomas Wallner
Enrolment Number: 1227490
Due Date: 4/4/14

Introduction

Self-organised criticality is a physical phenomenon in statistical physics, first discovered by Per Bak, Chao Tang and Kurt Wiesenfeld in 1987.[1] The concept behind self-organised criticality is that many dynamical systems, such as earthquakes, avalanches, starquakes, land formations and even stock market-dynamics, are inclined to arrange themselves in such a way that they tend to a critical point, hence, the name self-organised criticality. This report will investigate the self-organised criticality of sand piles by the means of computer simulation.

A pile of sand is constructed by adding grains of sand onto a flat surface at random positions. At some point in time, the system will reach its critical value and become unstable resulting in an avalanche. The size of the avalanche can vary greatly, from a small perturbation to a rearrangement of the entire pile of sand. Theory states that these phenomena exhibit power laws:

$$D(s) = s^{-\alpha}$$

Where D(s) the is frequency of avalanche sizes s. This Equation can be linearised to obtain α:

$$\ln(D(s)) = -\alpha \ln(s)$$

In the event that α≈1, the relationship is known as Pink noise or 1/f noise. Additionally, this relationship is valid for the distribution of avalanche duration:

$$D(t) = t^{-\alpha}$$
$$\ln(D(t)) = -\alpha \ln(t)$$

Where D(t) denotes the frequency of avalanches of duration t.

Model

In this simulation a 25 by 25 grid will be created, where each cell contains a given number of sand grains, less than the critical value (k). As the simulation begins, grains will be dropped one at a time on a random cell. When a cell reaches its critical value (k) the grid will be updated using the following rules:

$$If \quad cell(i,j)^n >= k \quad then$$
$$\circ \; cell(i,j)^{n+1} = cell(i,j)^n - 4$$
$$\circ \; cell(i \pm 1, j)^{n+1} = cell(i\pm, j)^n + 1$$
$$\circ \; cell(i, j \pm 1)^{n+1} = cell(i, j \pm 1)^n + 1$$

Where $cell(i,j)^n$ denotes the number of sand grains in row i and column j at the iteration n. In words, if a sand pile reaches its critical value, it loses 4 of its grains, which are distributed, one for each of its 4 direct neighbours.

[1] http://en.wikipedia.org/wiki/Self-organized_criticality

Methodology and OO design

In this simulation, there are four java classes. The following flow chart visualises the simplified interaction of all java classes.

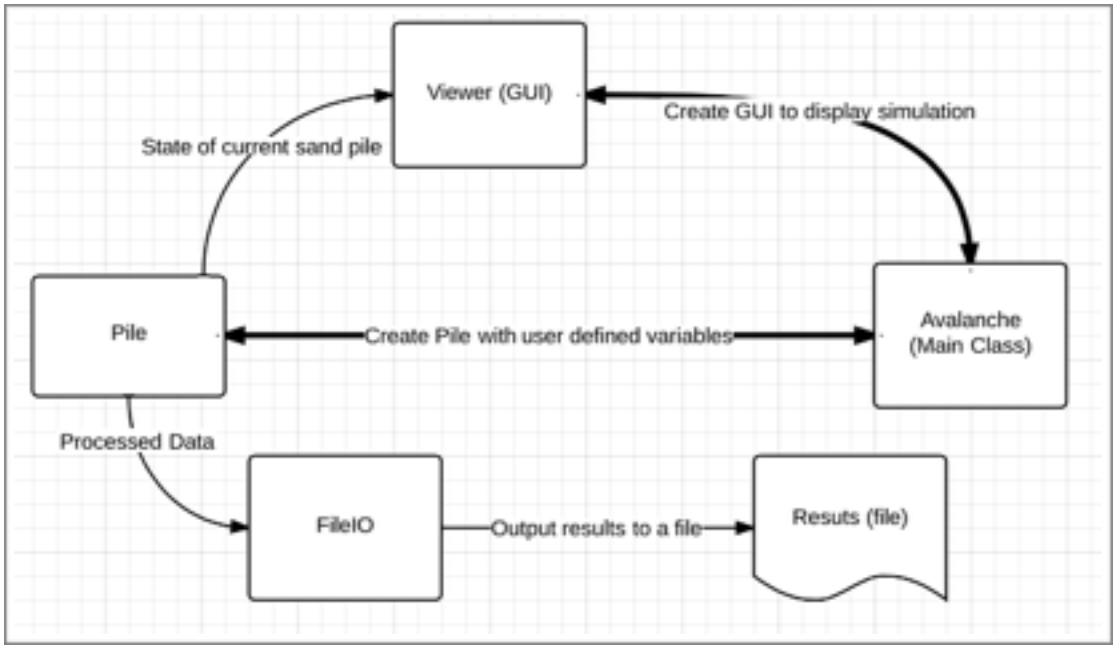


Figure 1 - Interaction chart of classes

This section will list all classes, along with their variables, constructors and methods. Then, a description outlining the purpose of each class and its design choices will follow.

Pile Class:

Variables	Type	Description
n	int	Sets the length and width of array (sand pile)
sand	int[ ][ ]	2D array holding the sand piles
sData	int[ ]	Collection of recorded data
sResult	int[ ]	Collection of processed data
sCount	int	Records data
critical	int	Critical value
stickiness	double	Probability of avalanche for a critical event
clumpsize	int	Determines the size of sand clumps added to sand pile
graphics	Graphics	Allows for access to Viewer class
viewer	Viewer	Allows for access to Viewer class

Constructor	Parameters	Description
Pile	n, critical, stickiness, clumpSize, viewer	Creates pile object with user defined paramters

Methods	Parameters	Return Type	Description
updatePile	int maxIterations	void	Adds a grain of sand to a random sand pile. Given a pile reaches its critical values, this method calls on the handleCritical method. Also, it records data on the frequency of avalanche sizes.
handleCritical	int i, int j	void	Handles the event of a critical value, in which case it applies the rules mentioned in the model section of report. This is a recursive method.
fillPile	none	void	Fills the grid with randomly generated number of sand grains for each sand pile. The randomly generated number must be between 0 and the critical value.
draw	none	void	Visualises the grid of sand piles by drawing a coloured rectangle (color depends on size of pile) for each sand pile
isCritical	int i, int j	boolean	Returns whether a sand pile of interest is critical
print	none	void	Prints the current arrangement of pile. Print method was used during coding for debugging
sMaxValue	none	int	Returns the maximum size of an avalanche that occurred
sProcess	none	int[ ]	Processes the Raw data so that it can be conveniently exported to microsoft excel for further processing and plotting

Note, the accessors and mutators are not listed.

The pile class contains the main meat of the code. First, it creates the grid of sand piles, in the form of a 2D array, which is most suited. It then updates a pile by adding one grain of sand to a random position at a time (updatePile method). In the event of a sand pile reaching a critical value, it calls on the recursive handleCritical method, meaning it calls on itself, in the event of another critical value occurring. On a side note, an assumption was made that any sand that is placed outside the grid by an avalanche is lost. During simulation, the pile class alters the Buffered Image in real time, so that the user can visualise the simulation in action. Additionally, it gathers raw data while updating piles, and then processes the data into a convenient format for later use.

FileIO Class:

Variables : none

Constructor	Parameters	Description
FileIO	none	Creates object which gives access to write method for main class

Methods	Parameters	Return Type	Description
write	int[]	void	Takes in the results from the simulation and writes them to a file, where they can be conveniently exported to an excel sheet for further data processing and graphing

The FileIO class is responsible for all file handling. Processed data is inputted and then outputted to the file ‘AvalancheResults’ via the write() method. This class is constructed for

simplicity and more elegant design. Alternatively, the implementation of a write method in the Pile class could have been undertaken, but this choice lacks style and does not make use of object oriented programming.

Viewer Class:

Variables	Type	Description
BI	BufferedImage	This is the actual image object that is inserted onto the JFrame. It holds an image of all coloured rectangles, which represent the sand piles.

Constructor	Parameters	Description
Viewer	none	This constructor instantiates the frame, which holds the buffered image.

Methods	Parameters	Return Type	Description
paint	Graphics	void	Standard paint method does the actual drawing of the buffered Image to the Graphics g.
getBI	none	Buffered Image	A simple accessor method. The Pile class can then access the Buffered Image and alter it during simulation.
drawImage	none	none	This method is used by the Pile class so that it can repaint() during simulation.

The Viewer class is responsible for the graphical user interface of the project. A Frame of size 500 by 500 pixels is created, an appropriate size to run the simulation. It makes use of a Buffered Image object to display the simulation live. The methods allow for the Pile class to access to Buffered Image and alter it in real time. The Buffered Image allows for smooth transitions of the rapidly changing GUI and is simple to code.

Avalanche Class:

The Avalanche class is the main class of this project, it brings together all other classes. First, it instantiates a viewer object from the Viewer class to create a frame. Next, a pile of user defined parameters is created by making use of the Pile class, and then updated 10000 times. Finally, it creates an array of integers to hold the processed data and then calls on the FileIO class to write the results to a designated file.

Improvements:

The code outputted ‘processed data’, in the sense that it produced the entire range of sizes of avalanches and their given frequency. However, since there are a lot of avalanche sizes (usually the larger ones) that don't have a single avalanche occurring, further processing was necessary. Also, the data had not been linearised yet. In hindsight, it would have been more efficient if the output of the code was fully processed. Furthermore, a graphical output of the code itself would eliminate the time needed to export the data into Microsoft excel. Time constraints did not allow for these modifications.

Results

The simulation was tested in a 25 by 25 grid, with a critical value of 5. The number of iterations was set to 10,000.

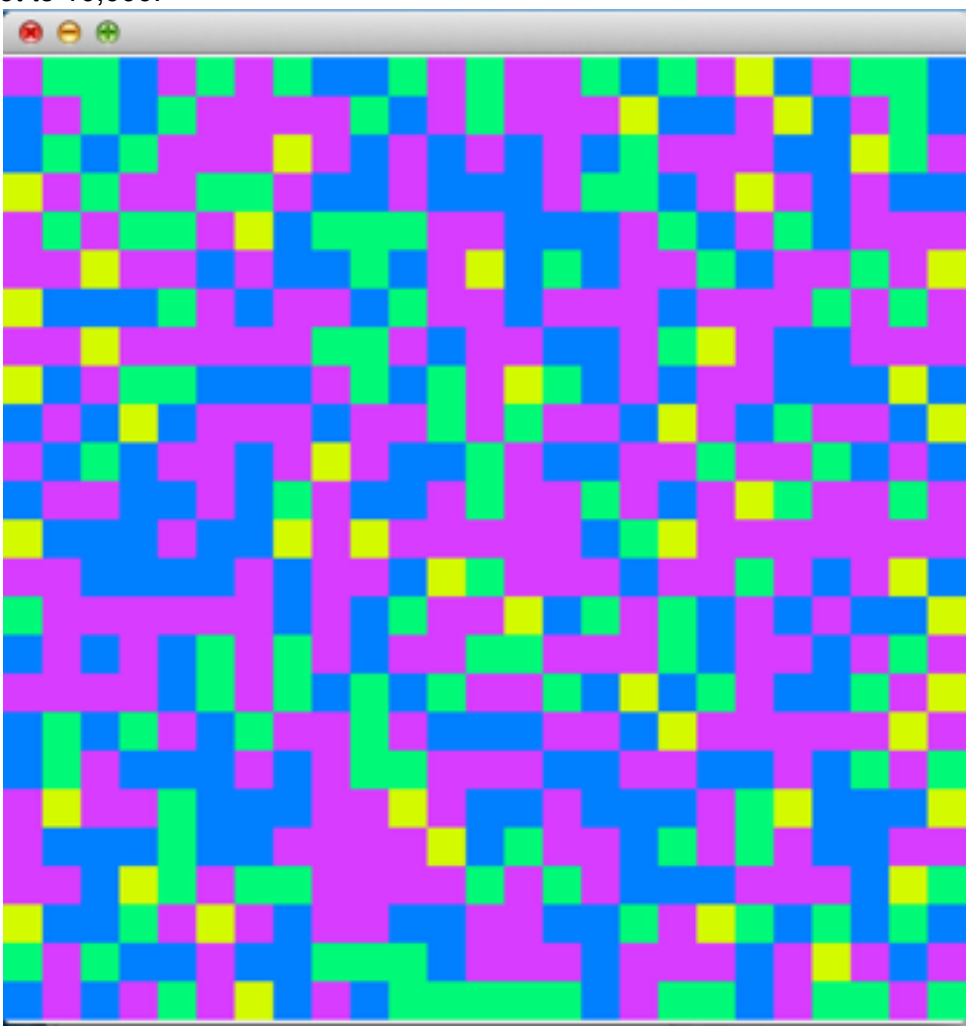


Figure 2 - Visualisation of simulation for testing

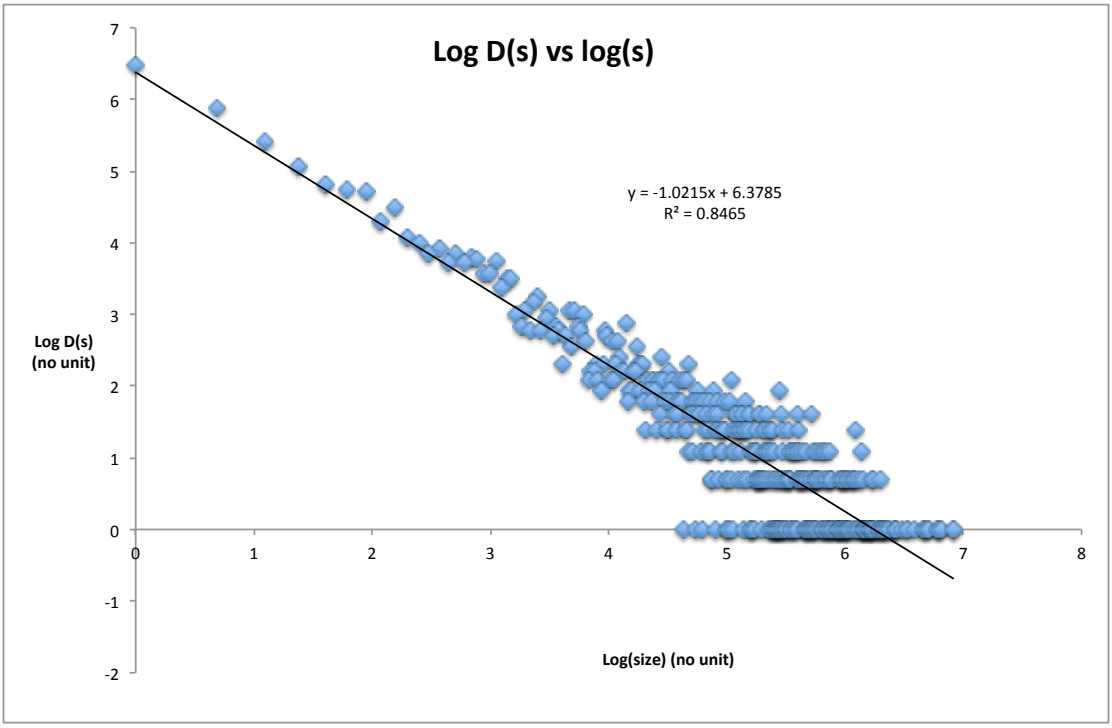
As seen in Figure 2, the the individual piles are colour coded, depending on how close the individual pile is to the critical value. In this specific test, purple corresponds to a sand pile one less than critical value.

Experiment 1

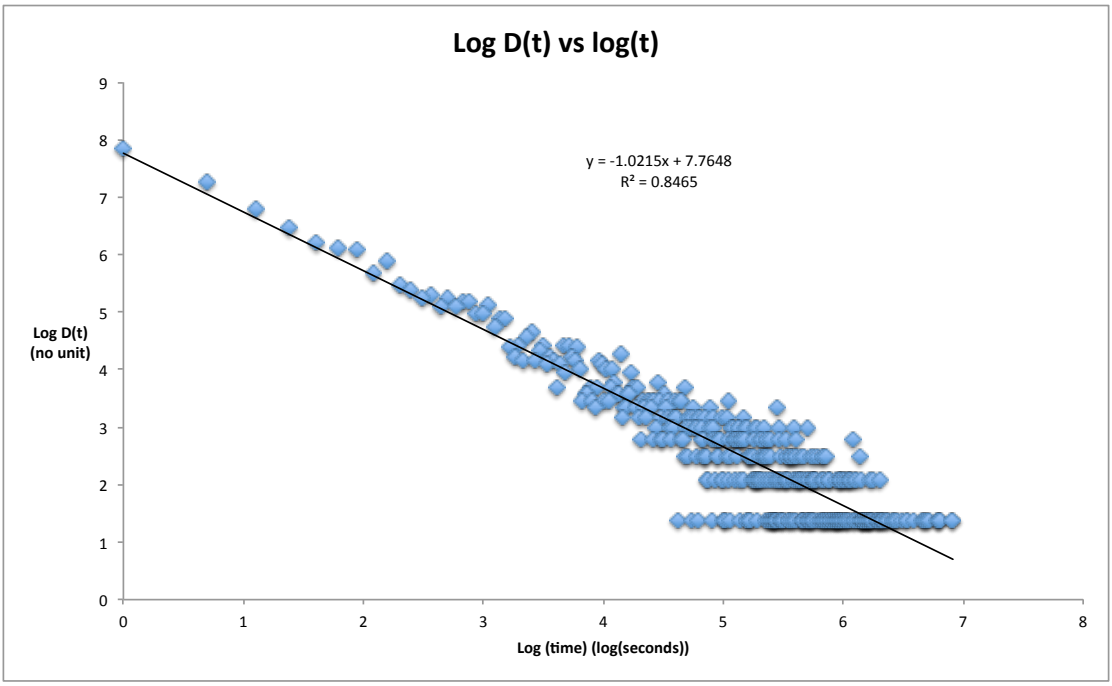
The aim of experiment 1 is to investigate the distribution of avalanche sizes D(s) and s for a range of critical values k. Additionally, a similar investigation for the duration of avalanches D(t) as a function of t is demonstrated.

To approach this problem, three simulations were carried out with the following parameters and results:

Critical value k= 5, 10000 Iterations

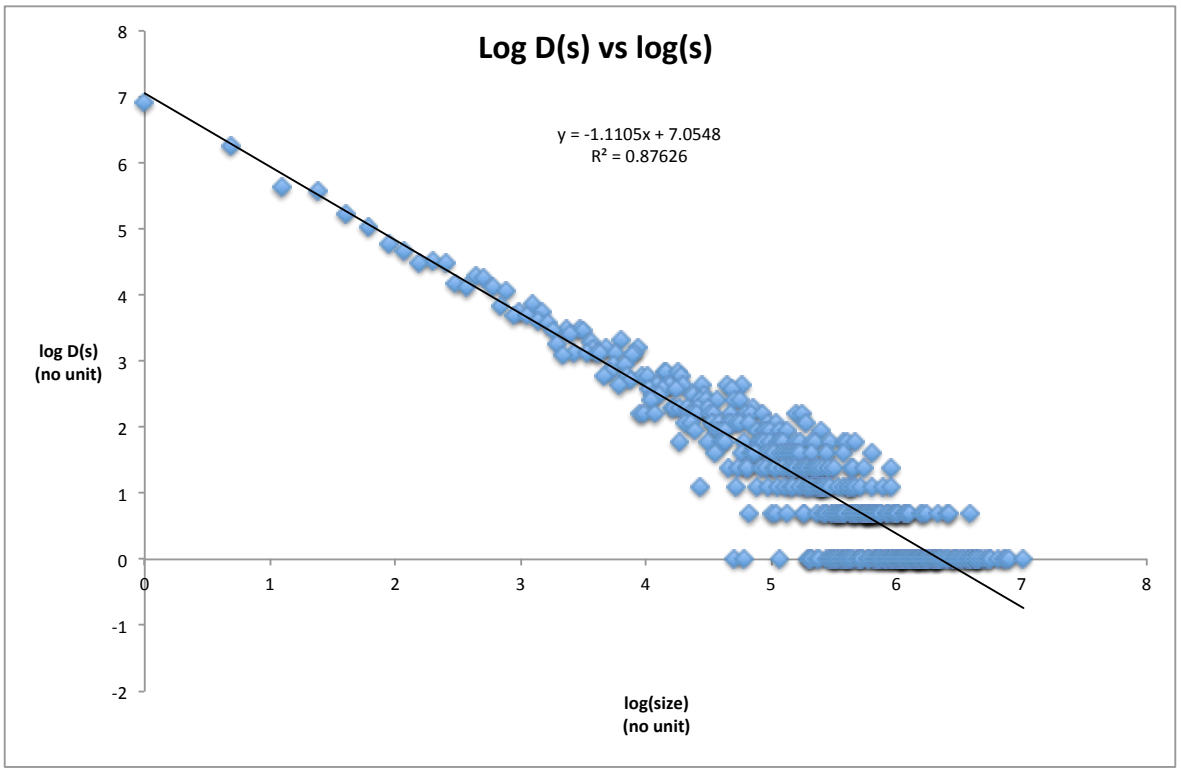


Graph 1 - Log D(s) vs log(s) - k=5

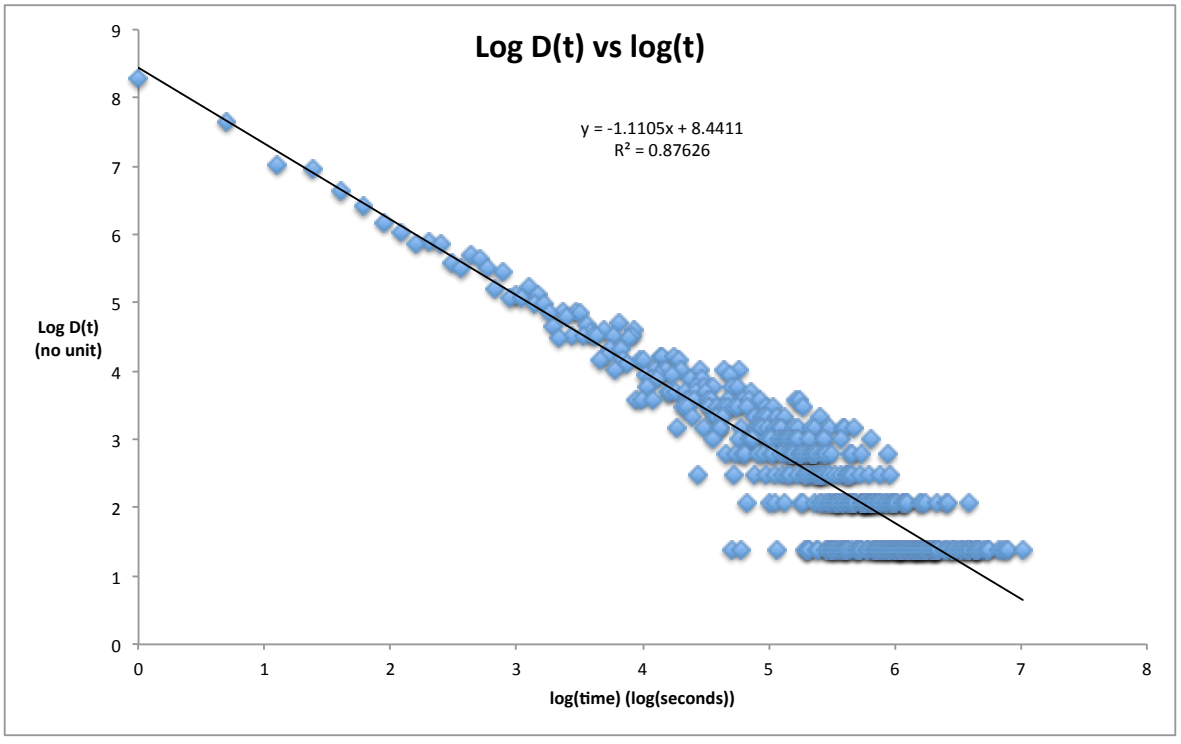


Graph 2 - Log D(t) vs log(t) - k=5

Critical value = 10, 15000 Iterations

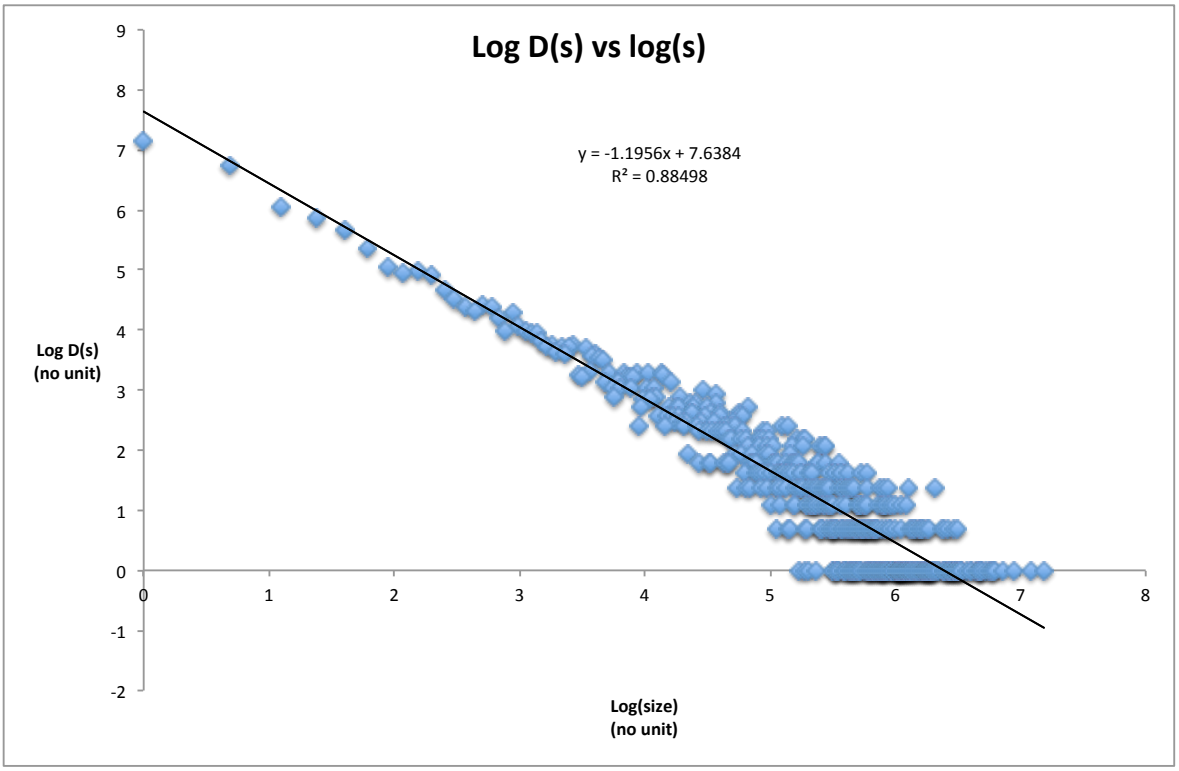


Graph 3 - Log D(s) vs log(s) - k=10

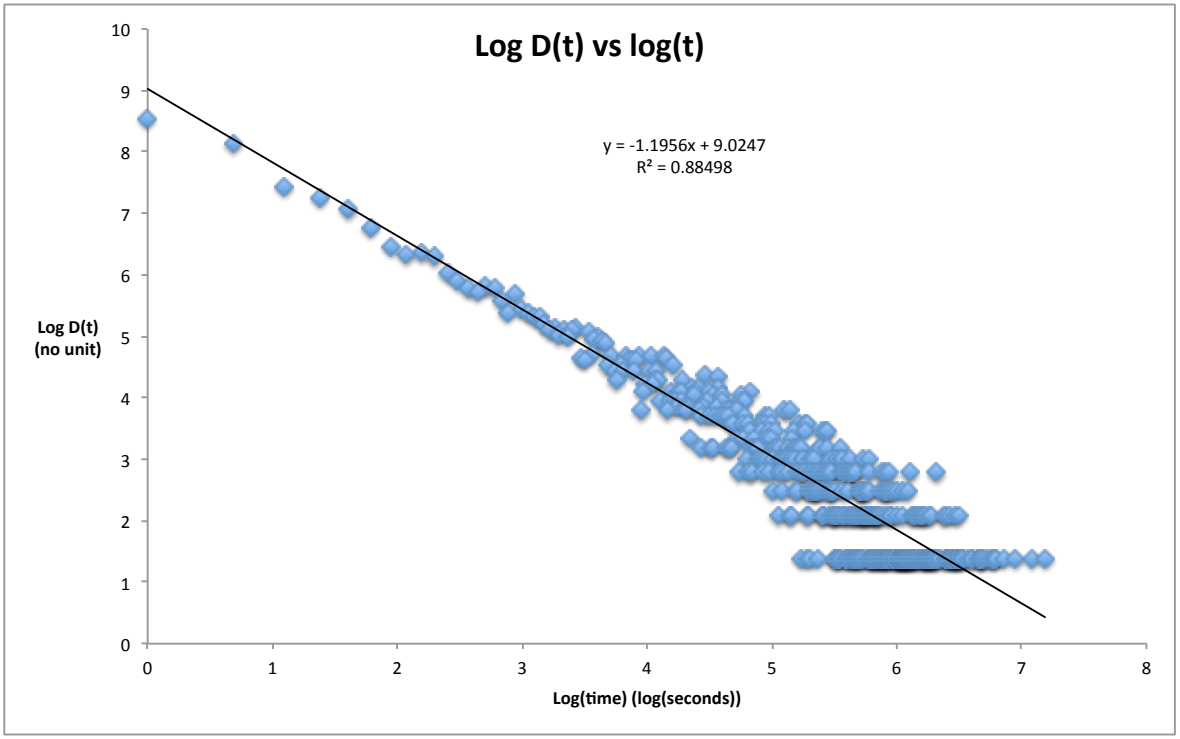


Graph 4 - Log D(t) vs log(t) - k=10

Critical value = 15, 20000 Iterations



Graph 5 - Log D(s) vs log(s) - k=15



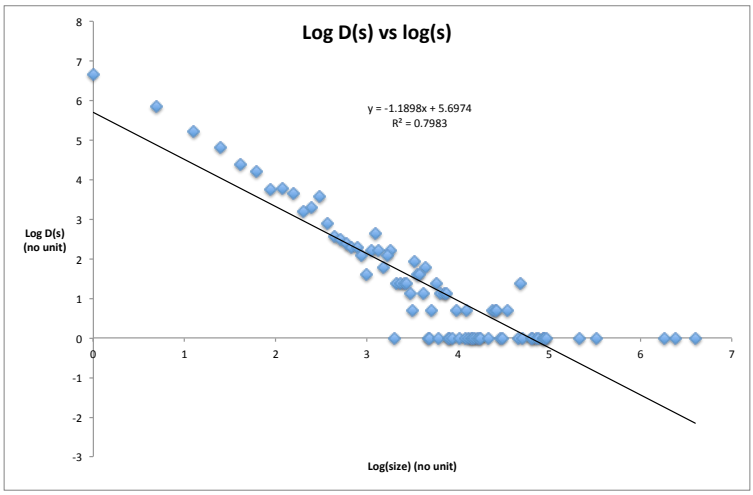
Graph 6 - Log D(t) vs log(t) - k=15

Experiment 2

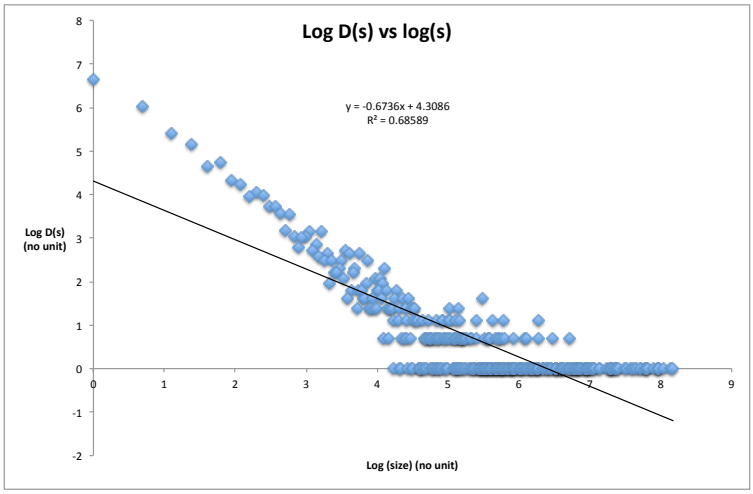
In experiment 2, ‘stickiness’ is added to the simulation. This means that if a sandpile reaches its critical value, there is a possibility for no avalanche to occur due to the stickiness.

I would like to note that I have coded a stickiness to ’s’, to correspond to a probability of an avalanche occurring of ‘1-s’. For example, a stickiness of 0.25 gives a 75% chance of an avalanche occurring.

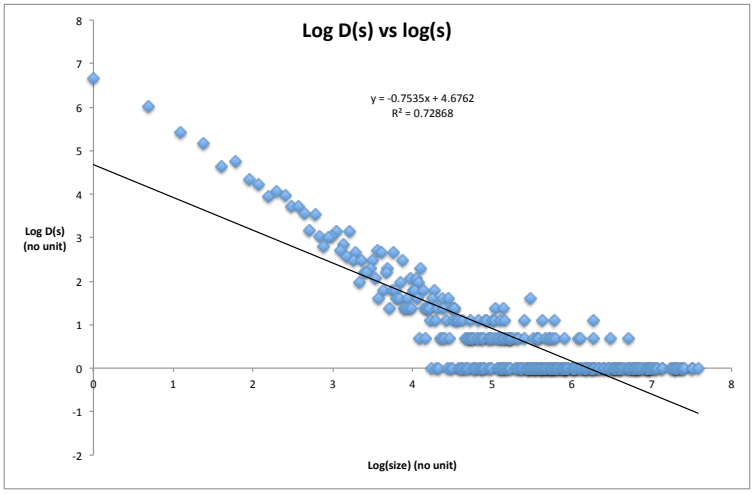
The following simulations were carried out to investigate stickiness:



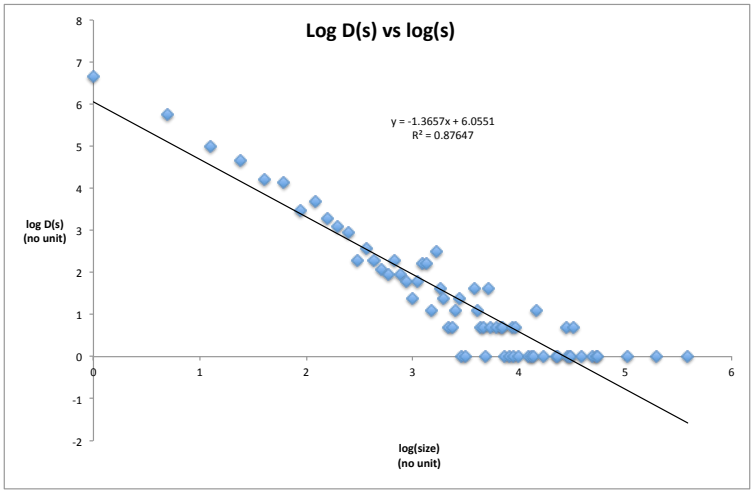
Graph 7 - Critical value = 5, Stickiness 0.25



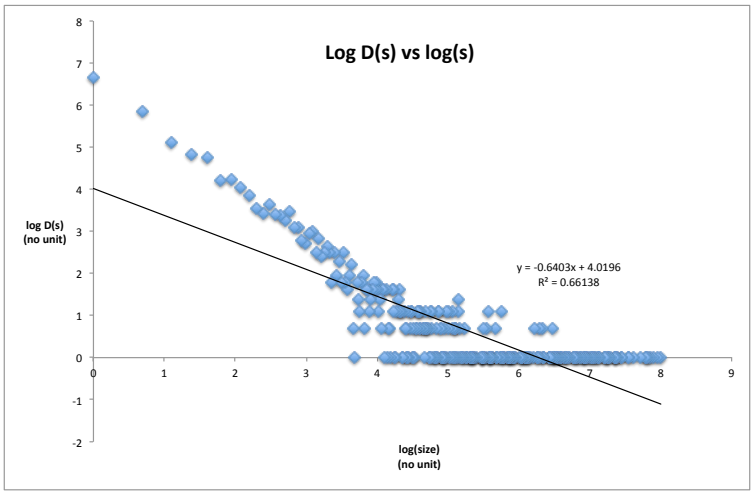
Graph 8 - Critical value = 5, Stickiness 0.50



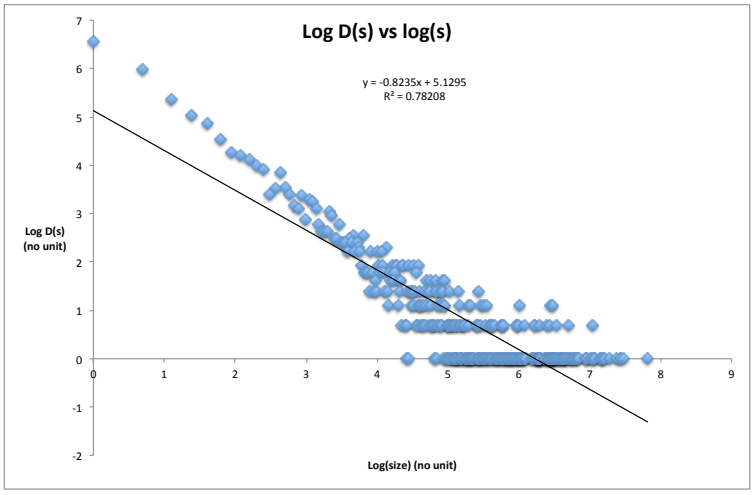
Graph 9 - Critical value = 5, Stickiness 0.75



Graph 10 - Critical value = 10, Stickiness 0.25



Graph 11 - Critical value = 10, Stickiness 0.50



Graph 12 -Critical value = 10, Stickiness 0.75

Discussion

The graphs 1, 3 and 5 for experiment 1 clearly depict the linear relationship in a logarithmic plot for the frequency D(s) of avalanche sizes s. This supports the hypothesis that the distribution of avalanche sizes follow an inverse power law. The values of α for k=5, k=10, k=15, are 1.0215, 1.1105 and 1.197, respectively. For all critical values α≈1, thus the relationship is known as 1/f or flicker noise. Evidently, regardless of the value of k, the hypothesis still holds true. There are no major sources of error.

For higher values of k, a higher number of iterations is required for the relationship to hold. Thus it deemed appropriate to increase the number of iterations by 5000 and 10000 for k=10 and k=15, respectively.

The relation holds in an identical fashion for the frequency D(t) of avalanche duration t. Note that the duration of one avalanche is four seconds for each individual avalanche, one second for each sand grain. For example if the total avalanche is of size 15, then the total duration of the entire avalanche is 60 seconds. Due to this fact, we should expect the same graph, except shifted. This means the values of α are the same! This is in fact what is depicted in the Graphs 2, 4 and 6. Thus, the hypothesis also holds true for the distribution of avalanche duration.

Question 1: What is the interpretation of alpa. Use alpha to predict probability of an avalanche occurring that will cover the whole of the grid.

Alpha is the critical exponent. In essence, alpha characterises the relationship of the frequency of avalanche sizes. Given that alpha is known, one may compute the probability of a particular avalanche size occurring.

A total avalanche size of 125 is needed to cover the entire grid of 625 (25*25) grid slots (An avalanche of size 1 covers 5 grid spaces). So, the probability of this event is:

$$125^{-1.0215} = 0.00721 = .72\%$$

This calculation is for a critical value of k=5. Note, this calculation does not take into account that come avalanches could (most likely) have overlapping grid points.

In hindsight, the calculation for the duration of an avalanche is not genuine, since the speed at which the grains fall depends heavily on the stage of the avalanche and the total avalanche duration. For instance, one might expect an avalanche to start very slowly and speed up. This was not taken into account in this simulation.

For experiment 2, the graphs 7-12 depict how the stickiness affects the distribution of avalanche sizes. It is evident that the hypothesis does not hold true anymore. There seems to be a trend that the smaller the stickiness factor, the larger the amount of avalanches of size 1 occur. Also, the maximum avalanche size is greatly affected. For the most part, the max avalanche size increases by a factor of 1-3. However, this is only after considering a couple values of stickiness. Further research should be done for a greater range of stickiness values, but given the time constraints, this is not possible. The graphs 10-12 show that the critical value has no effect on the outcome of the simulation.

Question 2 : Assuming that damp sand is stickier than dry sand, would you prefer to walk on dry or damp sand-dunes - and why?

The data states that larger avalanches occur when the sand is sticky or wet. Under the assumption that the damp sand-dune is not completely sticky and that only very large avalanches can be of danger, it makes sense to walk on dry sand-dunes, since their maximum avalanche size is much smaller.

Conclusion

The graphs 1-6 clearly demonstrate the linear relationship in a log-log plot for the distribution of avalanche sizes. No major sources of error are present. In all cases, the line of best fit is appropriate, thus the theory is supported. The values of α for k=5, k=10 and k=15, are 1.0215, 1.1105 and 1.197, respectively. This signifies that the relationship is indeed '1/f' or flicker noise. Furthermore, the relation holds in an identical fashion for the frequency D(t) of avalanche duration t. Identical values are found for α.

When stickiness is added to the simulation, the graphs 7-12 clearly show that the theory no longer holds. Instead the frequency of very small and very large avalanches increased. More time is needed to investigate the relationship of stickiness and distribution of avalanche sizes, as the data is not clear cut.