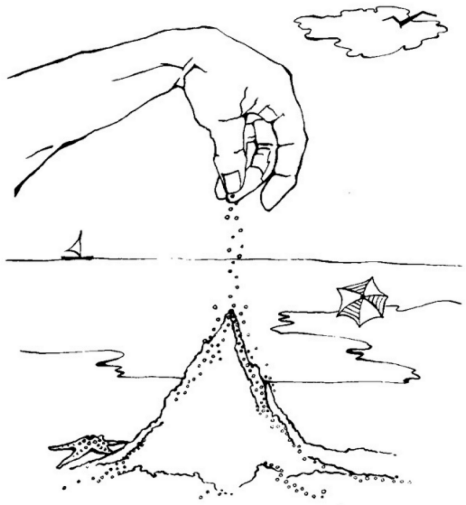
Sandpile model

Aldj Souleïman
Buccafurri Arnaud
Gaillot Solène
M1 Économétrie-Statistiques - Paris 1 Panthéon-Sorbonne

October 2025

Abstract

In this project, we will analyze the SandPile model, which is an example of a self-organized critical (SOC) system. Our goal here is to find the power-law distribution of avalanche sizes and its associated pink noise (1/f), which emerge naturally from the dynamics of the model.



Contents

1 Introduction

1.1 Presentation of the document

This document presents the sandpile model as an illustration of self organized criticality. In the introduction, we, first, explain the historical context with the article by Bak, Tang and Wiesenfeld (1987) and the key role of the $1/f$ noise in many natural systems. We then discuss exponential decay, power laws and their meaning in critical systems, followed by a short graph theory section that helps us understand the structure of the model.

The second part introduces the notion of self organized criticality and gives a simple example of it to show how this behavior appears in multiple contexts.

The third part explains the abelian structure that underlies the sandpile dynamics.

The fourth and fifth parts present the SAS code of the sandpile model, with a centered model and then with a random one. We analyze the avalanche size distributions and modelize the emergence of the $1/f$ noise.

Finally, we will propose an extension with the Burridge–Knopoff model which uses the same mechanisms in a seismic context.

1.2 Presentation of the article by Bak, Tang & Wiesenfeld (1987): Implications and applications

One of the great mysteries of statistical physics of complex systems, was the presence of 1/f noise, also called flicker noise (scintillation noise).

This noise is found in many phenomena and it can be observed, for example, in earthquakes with the dynamics of tectonic plates, in electrical signals between neurons, or in the evolution of prices on financial markets such as the CAC 40.

All these systems go through a phase of instability before self-organizing around a critical state. This is the phenomenon that Bak, Tang and Wiesenfeld wanted to understand by applying 1/f noise to their sandpile model. Their article marked a turning point because it offered a simple and concrete explanation of the Self-Organized Criticality and helped to better understand these complex phenomena.

The evolution of the CAC 40 illustrates the alternation between stability and sudden crises, characteristic of self-organized systems close to the critical state.

A few years later, the Indian physicist Deepak Dhar showed that these models follow an abelian dynamics which made it possible to better formalize it mathematically.

To explain their idea, Bak, Tang and Wiesenfeld start with a 1D model, made of pendulums. When there is a disturbance, the effect quickly disappears and the behavior remains quite simple and so not very interesting to analyze.

With the sandpile model, which is in 2D, a small instability can propagate and cause an avalanche that sometimes extends to a large part of the system. This phenomenon repeats itself until the system organizes itself around a critical state. This state is made up of many minimal stable states linked together. At that moment, there is no longer any scale of time or length, which is one of the main characteristics of SOC.

This repeated and scale-free avalanche behavior is found directly in the statistics of the system. To show this, it is necessary to compare a classical exponential decay to a power law, which is one of the clearest signatures of 1/f noise. This is what we will address in the next section.

1.3 Exponential Decay, Power Law and $1/f$ Noise in Self-Organized Systems

In a self-organized critical system, the distribution of avalanche sizes or durations follows a power law

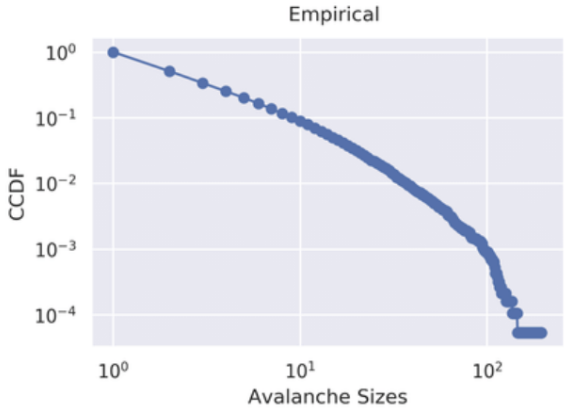
$$P(s) \sim s^{-\tau}$$

It is characterized by the absence of a characteristic scale, it means that small and large avalanches follow the same statistical structure. This behavior is typical of self-organized critical system, such as the Bak–Tang–Wiesenfeld sandpile model. Each added grain can trigger either a small or a large avalanche without a typical size. In theory, if the grid were infinite, we would observe an almost perfect straight line in a log–log regression. In practice, our grid has a finite size $n \times n$, and for this reason we observe at the end of the graph an exponential cut typical of exponential decay.

This cut can be written as

$$P(s) \sim s^{-\tau} e^{-s/s_c}$$

where s_c corresponds to the cutoff related to the system size. On the log–log graph, for example, we observe a linear part corresponding to the power law, then the break at the end which represents the exponential cutoff. This is called a truncated power law. It is a classical power law but limited by the maximum size an avalanche can reach in the system. So, the larger the grid, the further the cutoff.



The result from the SAS modeling will be presented later in section 5.

This distribution also translates into the frequency domain. Thanks to the Fourier transform, the global spectrum can be seen as the sum of contributions from all event durations

$$S(\omega) \propto \int_0^\infty \frac{T\, D(T)}{1 + (\omega T)^2}\, dT$$

If $D(T)$ follows a power law, we obtain

$$S(\omega) \sim \omega^{-\beta}$$

When β is close to 1, we recover $1/f$ noise. This result comes directly from the coexistence of different time scales in the critical system. The exponential cutoff simply shifts the spectrum to very low frequencies.

In summary, the truncated power law is a typical characteristic of a self-organized critical system. It reflects the behavior of the sandpile model and naturally leads to pink $1/f$ noise observed in many real-world phenomena.

1.4 Graph theory applied to the BTW model (SOC)

Graph theory was born in 1735 with Euler, at the crossroads of mathematics and computer science. It makes it possible to study graphs (here $n \times n$ grids) and the way objects are connected to one another. Graphs are made up of vertices (nodes) and edges (links).

In our case the graph is said to be undirected because there is a reciprocal link between vertices: when a vertex sends a grain to a neighbor, it can send one back later.

Topology here refers to the organization and overall structure of the graph. In our sandpile model it is regular because it is based on a square grid where each vertex is connected to its neighbors uniformly. This structural framework plays a central role in the propagation of avalanches.

Mathematically, a graph is defined by:

$$G = (S, A),$$

where G is the graph object, S the set of vertices and A the set of edges.

In our model:

$$S = \{(i, j) \mid 1 \leq i, j \leq n\},$$

where (i, j) represents the coordinates of a cell of the $n \times n$ grid.

The set A corresponds to pairs of neighboring vertices:

$$A = \bigcup_{(i,j) \in S} \big\{ \{(i,j), (k,\ell)\} \mid (k,\ell) \in V(i,j) \big\}.$$

Each element $\{(i, j), (k, \ell)\}$ represents an undirected edge, that is, a bilateral connection between two vertices. In other words, if (i, j) is linked to (k, ℓ) then (k, ℓ) is also linked to (i, j). It is a two-way road and not a one-way one.

The neighborhood $V(i, j)$ of a vertex (i, j) is given by:

$$V(i,j) = \{(i+1, j),\ (i-1, j),\ (i, j+1),\ (i, j-1)\},$$

where: $(i + 1, j)$ is the bottom neighbor, $(i - 1, j)$ is the top neighbor, $(i, j + 1)$ is the right neighbor and $(i, j - 1)$ is the left neighbor.

On the borders of the grid some of these neighbors do not exist (for example $(1, 1)$ has no neighbor above or to the left).

This lays the topological foundations (it describes who interacts with whom) needed to understand how the sandpile model works and the emergence of avalanches characteristic of SOC, which we will develop in detail later.

2 Self-Organized Criticality

2.1 Definition

Self-Organized Criticality (SOC) is a property of dynamical systems that naturally evolve toward a critical point. This means that the system organizes itself without external tuning of any parameters until it reaches a critical state (it is endogenous). SOC was an important discovery, although this property still lacks of a complete mathematical model.

Such systems self-organize when they are at the edge of stability (between order and chaos). In our case, the SandPile model becomes critical when a cell contains four grains of sand. At that point, an avalanche starts and spreads to adjacent cells. If a neighboring cell already had three grains, and now has four, the avalanche will continue (and thus, be bigger) until the system reaches stability again. The system becomes stable again when no cell contains four grains.

Self-organized critical models are particularly interesting near the critical point, where we can observe how the system transitions from instability to stability. These observations help us understand large-scale physical laws emerging from local interactions.

2.2 Exemple $n = 3$

In this exemple, we had +1 in center case.

		3		
	3	4	3	
		3		

		4		
	4	0	4	
		4		

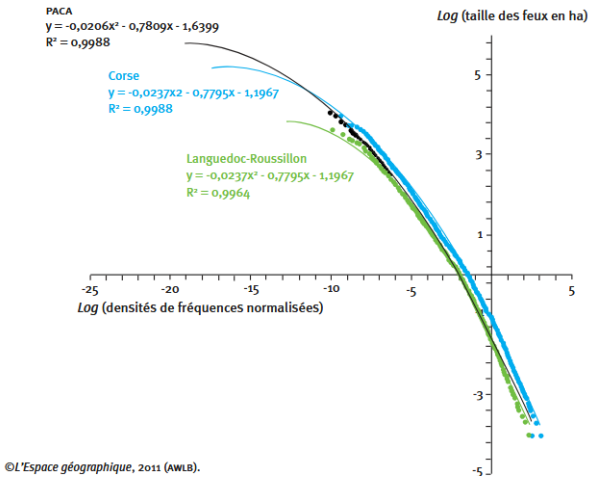
		1		
	2	0	2	
1	0	4	0	1
	2	0	2	
		1		

		1		
	2	1	2	
1	1	0	1	1
	2	1	2	
		1		

2.3 Forest Fire

Forest fires can be modeled as a self-organized critical system too. When a forest become too dense, fire acts as a vector of energy transfer, leading to the dissipation of energy. This phenomenon shows fractal properties, because when the forest condenses and trees grow too close to each other, the system becomes unstable. Eventually, the forest is destroyed by a fire, and then trees grow again.

We can observe flicker noise $(1/f)$ when plotting the logarithm of fire size against the logarithm of normalized frequency density. $1/f$ noise, will be discussed further in Section 5.



3 Abelian Geometry

3.1 Defintion

An abelian variety over a field $\mathbb{K}$ is an algebric group whose underlying algebric variety is connected and projective.
Let A be an abelian variety. It exists an internal composition law
$+ : A \times A \to A$ and a neutral element $0 \in A$ such that $(A, +)$ forms an abelian group.
An abelian group is a commutative group (i.e. $\forall B, C \in (A, +), B + C = C + B$.
Example: $y^2 = x^3 + ax + b$ (an elliptic curve) is an abelian variety when $\mathbb{K} = \mathbb{Q}, \mathbb{R}$ or $\mathbb{C}$.

Projective and Euclidean geometries are different. In projective geometry, at infinity, two distinct straight lines converge to the same point.

Abelian geometry is connected. A topological variety is connected if it is made of a single piece.
A non-oriented graph (V, E) is connected if $\forall u, v \in V$, $\exists\ c(u, ..., v)$.
In abelian geometry, the variety must be connected. A geometric variety X is connected if it cannot be decomposed into two subsets U, V like $U \cup V$ open, $\neq \emptyset$ and $U \cap V$ (This is the topological definition of connectedness).
Therefore, our algebric varieties are both projective and connected.

3.2 Critical abelian group

Let $G = (V, E)$ be a finite graph, and let $S \in V$ be the central vertex. For each vertex $v \in V$, let $c(v)$ be the number of grains.
A c configuration is stable if $c(v) < deg(v)$ for $v \neq s$ otherwise, the adjacent vertex to v get $deg(v)$ grain until stability.
A configuration c is said to be stable if $c(v) < deg(v)$ for all $v \neq s$. otherwise, the adjacent vertices of v each receive $deg(v)$ grain until the system becomes stable.

Let S be the set of all stable configurations, and let $\oplus_S$ be a commutative and distributive law.
For $c_1, c_2 \in V$, :

$$\oplus_S : c_1, c_2 \longrightarrow \text{stability}(c_1 + c_2)$$

Let $(S, \oplus_S)$ be a finite abelian group, we note $K(G)$ the critical group of the graph.

The Laplacian matrix L is defined as follows:

$$L = (L_{i,j})_{1 \leq i,j \leq n} \qquad (L_{i,j}) = \begin{cases} deg(v_i) & \text{if } i = j \\ -x_{i,j} & \text{if } i \neq j \text{ and } v_{i,j} \text{ the number of edge between } v_i \text{ and } v_j \\ 0 & \text{else} \end{cases}$$

We have the relation :

$$K(G) \cong \mathbb{Z}^{|V|-1} / Im(L)$$

- $K(G)$ is the critical group (also called the sandpile group) of the graph G.
- $|V| - 1$ is the number of vertices, excluding the central vertex s
- $Im(L)$ is the image of the Laplacian matrix, representing the redistribution of sand.

The critical abelian group is essential for modeling the sandpile model. Abelian varieties are also used in mathematical research, for instance in the study of zeta functions $\zeta(x)$.

4 SandPile Model (centered)

In order to illustrate the concepts presented above, we model a matrix of dimension 201×201. This matrix represents a sand plate on which a grain of sand is dropped onto the central cell at the beginning of each iteration.

When a cell contains 4 grains of sand (or more), the system becomes unstable and reacts in such a way that it naturally returns to a critical state (at the limit between stability and instability). This reaction is materialized by a "sliding" of the cell, the 4 grains present on it slide equally onto its neighboring cells, thus adding one grain of sand to the cell on the right, the left, the top, and the bottom of the sliding cell.

This behavior refers to self-organized criticality (SOC). The system naturally (without external tuning of any parameter) reaches a stationary critical state, resulting from the balance between the addition of a grain (which tends to make the system unstable) and the collapses (which dissipate this instability).

It is important to note that the system returns to a stable state when no cell in the matrix contains 4 or more grains of sand. Consequently, the addition of a single grain of sand to the central cell may trigger a chain reaction of varying magnitude, with several cells becoming unstable. This phenomenon is referred to as an avalanche. An avalanche is characterized both by the number of cells that have slid during the process and by its number of waves (the addition of a grain causes the central cell to slide = first wave, this sliding may in turn cause other cells to slide = second wave, ...).

The goal of our model is to reproduce the power-law distribution that characterizes the size of the avalanches. This distribution illustrates the absence of a typical avalanche size (very small and very large avalanches may occur according to a scale-free law). Furthermore, we observe a 1/f power spectrum (or pink noise), indicating that the fluctuations are correlated across all temporal scales (there is no dominant frequency). Both phenomena express the absence of a characteristic scale, in space (avalanche size) as well as in time (avalanche duration or frequency).
Here, we explain how our SAS code works, which allows us to obtain these results.

```
proc iml;

m = 21;
sand = j(m, m, 0);
center = ceil(m/2);
```

We begin by initializing the sand plate by creating a 201×201 matrix filled with zeros, and then we compute its center in order to identify the central cell later on.
Next, we initialize two empty matrices, respectively:
- Avalanche_Time (or t), which stores the number of waves required for an avalanche to end before the sandpile returns to a stable state. Here, we consider the number of waves as the "time" or "duration" of an avalanche.
- Avalanche_Size (or N), which stores the number of cells that have undergone a slide during an avalanche, that is, its size.

The first loop determines the number of grains (and thus the number of iterations) that we want to drop onto the central cell in our model. At the beginning of each new iteration, we reset the variables time and unstable to zero, which allows us to compute, for each avalanche, its duration and its size respectively.

```
do i = 1 to 120;
sand[center,center] = sand[center,center]
+ 1;

time = 0;
unstable = {};
```

```
do while(max(sand) >= 4);
affected = loc(sand >= 4);

if ncol(affected) > 0 then do;

time = time + 1;
sand[affected] = sand[affected] - 4;
```

Inside this first loop, we include a second one that stops when there are no longer any cells with a value greater than or equal to 4.
First, we locate the positions of the unstable cells in a variable called affected. If such cells exist, we add 1 to the time variable to count the number of waves needed for the avalanche to stabilize, and we record the affected cells in the matrix unstable.
The affected cells lose 4 grains, which "slide" toward their neighboring cells.

We then perform the redistribution of these grains to the neighboring cells of each unstable cell.
We start with the left cell (affected – 1). If it is within the matrix, the index is not divisible by the matrix dimension (its remainder is not zero), then it receives one additional grain.
We proceed in the same way for the right cell (affected + 1).

```
left = affected - 1;
if mod(left, m) ^= 0 then do;
sand[left] = sand[left] + 1;
end;

right = affected + 1;
if mod(right-1, m) ^= 0 then do;
sand[right] = sand[right] + 1;
end;
```

```
up = affected - m;
if up > 0 then do;
sand[up] = sand[up] + 1;
end;

down = affected + m;
if down<=m*m then do;
sand[down] = sand[down] + 1;
end;
end;
end;
```

Finally, we do the same for the cells located above and below each unstable cell.
For the upper cell (affected – m), we simply check that its index is greater than 0 to ensure it belongs to the matrix before adding one grain to it.
For the lower cell, we check that its index is less than or equal to the last cell of the matrix (m×m).
If a neighboring cell lies outside the matrix, the grain that was supposed to slide onto it is considered to leave the system.
We end the second loop here; it will repeat if the sliding of unstable cells creates new unstable cells, and this continues until the entire matrix becomes stable again (no cell has a value greater than or equal to 4).

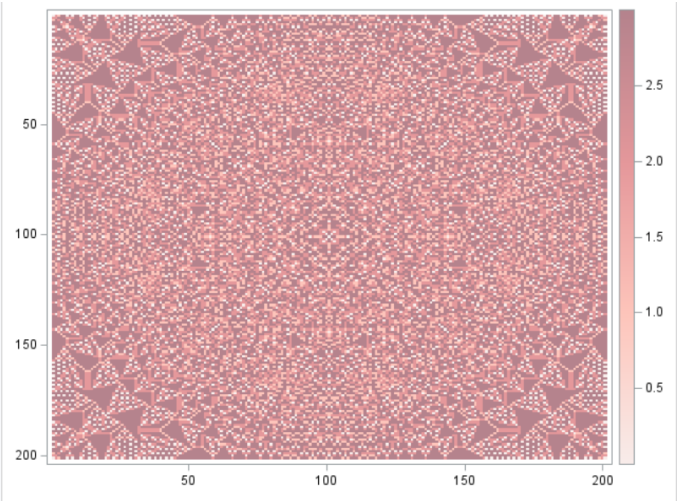
Finally, we store the data collected for each avalanche.
For the avalanche size, we are no longer interested in the precise locations of the affected cells, but only in their number. We add this number to the matrix that records the sizes of all avalanches.
Similarly, the avalanche duration is added to the corresponding matrix.
We then terminate the first loop, allowing the next grain to fall and the process to start again for a new iteration.

```
unstable = ncol(unstable);
avalanche_Time = avalanche_Time //
time;
avalanche_Size = avalanche_Size // un-
stable;
end;
```

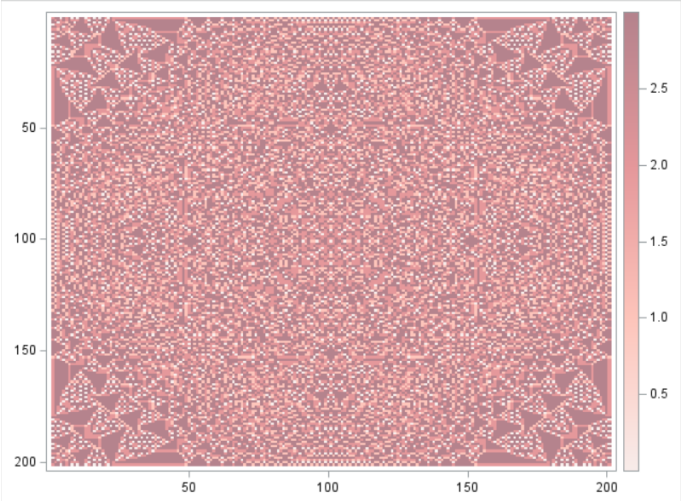
```
call heatmapcont(sand)
colorramp= { 'CXF8EDEB'
'CXFEC5BB' 'CXE5989B' 'CXB5838D'}
displayoutlines=0;
```

Finally, we display our matrix using colors: the more grains there are on a cell, the darker it appears.

For a 201×201 matrix, after 500 million grains have been dropped at the center, we obtain the following result:



For a 201×201 matrix, after 750 million grains have been dropped at the center, we obtain the following result:



The colors highlight the deterministic and self-organized nature of the model, but they also reveal the Abelian geometry of the system. Indeed, the sandpile model is said to be Abelian because the order in which the unstable cells slide has no influence on the final state of the system once stability is reached. The result is said commutative, if two unstable cells stabilize in one order or the other, the final configuration of the matrix will be exactly the same.

5 SandPile Model (regression)

We want to reproduce the curve illustrating the distribution of avalanche sizes. To do this, we now need the grains of sand, which until now were dropped onto the center of the sandpile at the beginning of each iteration, to fall randomly onto a cell of the matrix. We make sure that the selection of the cell follows a uniform distribution. And, to obtain a clearer illustration of the phenomenon, we reduce the matrix size to 21×21 and the number of grains to 50,000.

We replace the beginning of the code with the following new code:

```
xaxis = ceil(m*randfun(1, "Uniform"));
yaxis = ceil(m*randfun(1,"Uniform"));

sand[xaxis,yaxis] = sand[xaxis,yaxis] + 1;
```

In order to use the two matrices storing the avalanche data (avalanche_Size and avalanche_Time) outside of the IML procedure, we convert them into a dataset, while removing all iterations that did not trigger an avalanche (avalanches_size = 0).

```
create avalanche_Size from avalanche_Size[colname=
                                        "size_of_an_avalanche"];
append from avalanche_Size;
    append from avalanche_Size;
close avalanche_Size;
create avalanche_Time from avalanche_Time[colname=
                                    "duration_of_an_avalanche"];
    append from avalanche_Time;
close avalanche_Time;
quit;
data avalanche_Size;
    set avalanche_Size;
    where size_of_an_avalanche > 0;
    observations = _N_;
run;
```

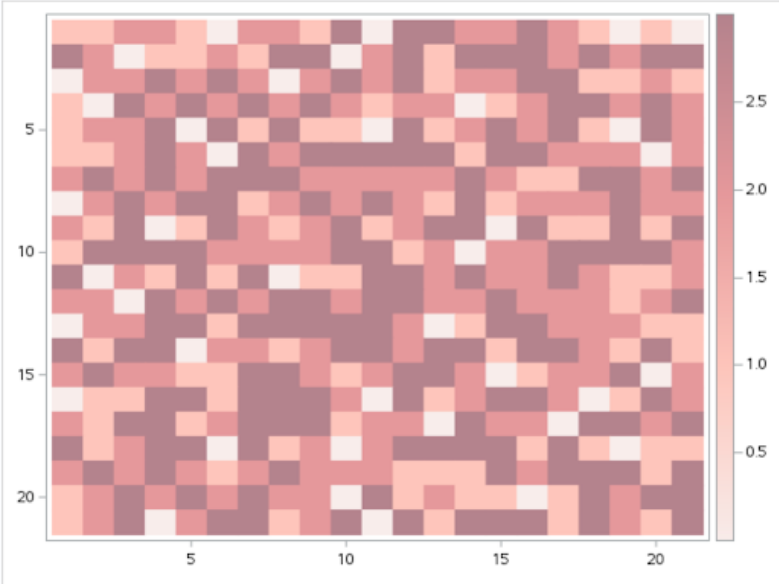
```
data avalanche_Time;
    set avalanche_Time;
    where duration_of_an_avalanche > 0
run;
```

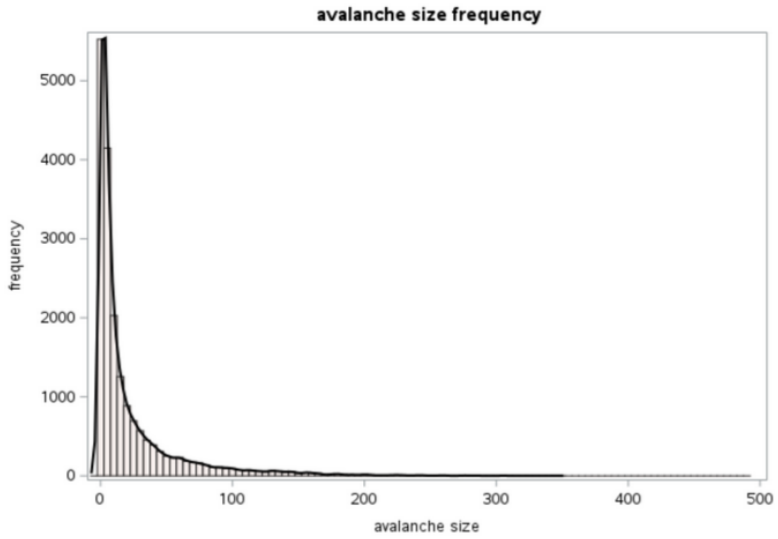
To do the regression, we need the frequency of occurrence for each avalanche size. We start by creating a graph that illustrates the distribution of avalanche sizes using a kernel density curve.

```
proc sgplot data=avalanche_Size;
histogram size_of_an_avalanche / nbins=100 scale=count transparency=0.2

fillattrs=(color=CXF8EDEB);

density size_of_an_avalanche / type=kernel lineattrs=(color=black thickness=2);
xaxis label="avalanche size";
yaxis label="frequency";
title "avalanche size frequency";
run;
```





We observe that the larger the avalanche, the smaller its occurrence.

We now begin the regression on the scatter plot.

```
proc freq data=avalanche_size noprint;
    table size_of_an_avalanche /
    out = frequence_of_avalanches_size;
run;
data frequence_of_avalanches_size;
    set frequence_of_avalanches_size;
    log_size_of_an_avalanche = log(size_of_an_avalanche);
    log_percent = log(Percent);
    drop Count;
run;
```

First, we save the frequencies in a dataset (frequency_of_avalanche_sizes) so that they can be manipulated. We decide to focus only on the frequencies expressed as percentages.
The regression will be in the logarithmic form, since the power-law relationship has the form:

$$Y = ax^b$$

We transform it in such a way that we can determine the values of the coefficients:

$$\ln(y) = \ln(a) + b\ln(x)$$

Intuitively, we thought that a standard ordinary least squares (OLS) regression would be the most appropriate, such as:

$$\text{log\_percent} = a + b\,\text{log\_size\_of\_an\_avalanche}$$

However, after plotting the regression line on the scatter plot, we notice that the curve does not follow the expected power-law shape. Indeed, the OLS line does not encompass the scatter plot; it merely passes through it.

Moreover, our system does not evolve on an infinite grid, which means that the scatter plot is not perfectly linear. At the end of the plot, we observe a cutoff where the data transition from a linear (power-law) behavior to an exponentially decaying one. This deviation occurs because our grid is finite.
Two possible approaches can be considered:
- We can perform the regression without taking into account the exponentially decaying part of the scatter plot, in order to recover the pure power-law behavior.
- Or we can use a power-law distribution with an exponential cutoff(often applied in network physics).

We choose the second approach and use the power-law with exponential cutoff model:

$$Y = a\,x^{b_1}\,e^{b_2 x}$$

$$\iff \ln(Y) = \ln(a) + b_1\ln(x) + b_2 x$$

$$\iff \text{log\_percent} = a + b_1\,\text{log\_size\_of\_an\_avalanche} + b_2\,\text{size\_of\_an\_avalanche}$$

```
proc reg data=frequence_of_avalanches_size outest=parametres_ regression noprint;
    model Percent = log_size_of_an_avalanche size_of_an_avalanche / stb;
    output out=predictions_regression
    p=yhat ;
run;
```

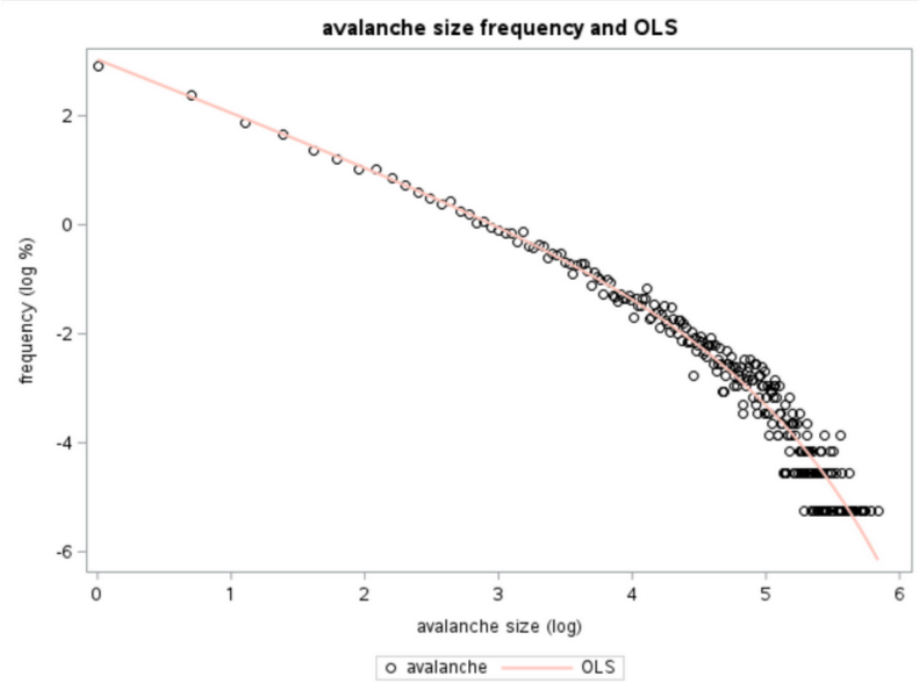
After coding the regression we found these parameters for a 21×21 matrix and 50,000 grains :

$\text{log\_percent} = \mathbf{3.04} - \mathbf{0.96}\,\text{log\_size\_of\_an\_avalanche} - \mathbf{0.01}\,\text{size\_of\_an\_avalanche}$

The larger our grid becomes (as it tends toward infinity), the closer the coefficient $b2$ approaches 0, eventually approaching a pure power-law behavior with a coefficient $b1$ close to -1.

We then display the graphical representation of its power-law distribution:

```
proc sgplot data=regression;
scatter x=x y=y / markerattrs=(symbol=circle color=black) legendlabel="avalanche";
series x=x y=yhat / lineattrs=(color=CXFEC5BB thickness=2) legendlabel="OLS";
xaxis label="avalanche size (log)";
yaxis label="frequency (log %)";
title "avalanche size frequency and OLS";
run;
```



We clearly observe the characteristic power-law behavior of avalanche sizes in the first part of our graph. This confirms that the system exhibits scale-free dynamics, characteristic of self-organized criticality (SOC).

Plus, as $b_1 \approx -1$, the power-law distribution of avalanche sizes is closely related to the 1/f power spectrum (pink noise) in the temporal evolution of the system. Since avalanches occur over an important range of sizes and durations, the superposition of these events produces fluctuations that are correlated across different time scales. So, the system shows long temporal correlations (there is no dominant frequency, and the activity follows a scale-free pattern in time).

Together, the power-law distribution of avalanche sizes and the 1/f temporal spectrum provide evidence of the self-organization criticality of the sandpile model.

We can also do the same by using the waves (avalanche duration) instead of the affected cells (avalanche size). We will obtain similar results, although they will be slightly less visually pronounced.

6 Extansion of sandpile model

For the extension of the sandpile model we chose the Burridge–Knopoff model which illustrates well the dynamics of a SOC system but applied to the seismic domain with a slow and fast dynamic. To have the best possible analogy with the sandpile model we will focus only on the discrete version of the BK and not on its continuous version.

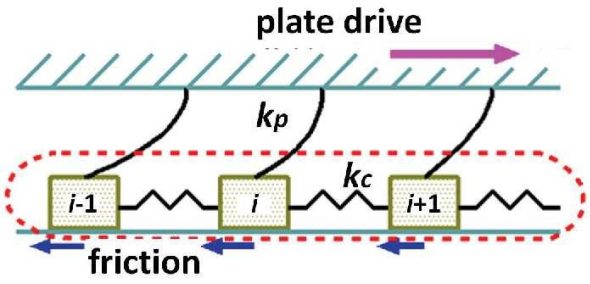


Diagram of the Burridge–Knopoff model in 1D.

We start from a site i representing a block connected to an upper plate by a spring k_p and to its neighbors $i-1$ and $i+1$ by springs k_c. The plate moves slowly and increases the stress on the block as when a grain falls on a cell of the sandpile. Each block is immobile due to friction until the stress exceeds a certain threshold. When this threshold is reached the block slides and transfers part of the stress to its neighbors through the springs like an unstable cell that topples and redistributes its grains.

Even if the dimension here is 1D and not 2D the dynamic structure is the same : the loading is slow, there is a local threshold, the stress is redistributed to neighbors, avalanches of different sizes can appear, and when the rupture reaches the boundary of the system, part of the energy leaves the system like the grains at the edges of the sandpile.

This discrete version of the BK reproduces the same characteristics as the BTW model with a power law distribution of avalanches. It also generates a 1/f pink noise in the frequency domain because the superposition of events of different sizes and durations creates fluctuations correlated at all time scales.

7 Bibliography

– SOC course, University of Frankfurt

– Abelian sandpile model, Rossin (École Polytechnique)

– Abelian sandpile model, Wikipedia (EN)

– Student project, Sandpile Model (University of Frankfurt)

– SageMath, Sandpile model tutorial

– Rosetta Code, Abelian sandpile implementation

– Scientific article, Collapse Disc Nga VR (PDF)

– Graph theory, Wikipedia (FR)

– Power law, Wikipedia (EN)

– Power law vs exponential, ResearchGate

– Historical chart of CAC 40, Wikipedia (media)

– Les incendies de forêt méditerranéens : un système critique auto-organisé modélisé en géographie

8 Annexe

```
proc iml;
m = 21;
sand = j(m, m, 0);
avalanche_Time = {};
avalanche_Size = {};

do i = 1 to 50000;
xaxis = ceil(m*randfun(1, "Uniform"));
yaxis = ceil(m*randfun(1,"Uniform"));
sand[xaxis,yaxis] = sand[xaxis,yaxis] + 1;

time = 0;
unstable = {};

do while(max(sand) >= 4);
affected = loc(sand >= 4);

if ncol(affected) > 0 then do;

time = time + 1;
sand[affected] = sand[affected] - 4;
unstable = unstable || affected;

left = affected - 1;
if mod(left, m) ^= 0 then do;
sand[left] = sand[left] + 1;
end;

right = affected + 1;
if mod(right-1, m) ^= 0 then do;
sand[right] = sand[right] + 1;
end;

up = affected - m;
if up > 0 then do;
sand[up] = sand[up] + 1;
end;

down = affected + m;
if down<=m*m then do;
sand[down] = sand[down] + 1;
end;
```

```
end;
end;

unstable = ncol(unstable);
avalanche_Time = avalanche_Time // time;
avalanche_Size = avalanche_Size // unstable;
end;

call heatmapcont(sand)
colorramp= { "CXF8EDEB" "CXFEC5BB" "CXE5989B" "CXB5838D"}
displayoutlines=0;

create avalanche_Size from avalanche_Size[colname={"size_of_an_avalanche"}];
append from avalanche_Size;
close avalanche_Size;
quit;

data avalanche_Size;
set avalanche_Size;
where size_of_an_avalanche > 0;
observations = _N_;
run;

proc freq data=avalanche_size noprint;
table size_of_an_avalanche / out = frequence_of_avalanches_size;
run;

data frequence_of_avalanches_size;
set frequence_of_avalanches_size;
log_size_of_an_avalanche = log(size_of_an_avalanche);
Percent = log(Percent);
drop Count;
run;

proc reg data=frequence_of_avalanches_size
outest=parametres_regression noprint;
model Percent = log_size_of_an_avalanche size_of_an_avalanche / stb;
output out=predictions_regression
p=yhat;
run;

data parametres;
set parametres_regression(
```

```
keep=Intercept log_size_of_an_avalanche size_of_an_avalanche
rename=(
Intercept = constante
log_size_of_an_avalanche = beta1
size_of_an_avalanche = beta2
)
);
run;

data regression;
set predictions_regression(
keep=log_size_of_an_avalanche Percent yhat
rename=(
log_size_of_an_avalanche = x
Percent = y
yhat = yhat
)
);
run;

proc print data=parametres;
run;

proc sgplot data=avalanche_Size;
histogram size_of_an_avalanche / nbins=100 scale=count transparency=0.2
fillattrs=(color=CXF8EDEB);
density size_of_an_avalanche / type=kernel lineattrs=(color=black thickness=2);
xaxis label="avalanche size";
yaxis label="frequency";
title "avalanche size frequency";
run;

proc sgplot data=regression;
scatter x=x y=y / markerattrs=(symbol=circle color=black)
legendlabel="avalanche";
series x=x y=yhat / lineattrs=(color=CXFEC5BB thickness=2)
legendlabel="OLS";
xaxis label="avalanche size (log)";
yaxis label="frequency (log %)";
title "avalanche size frequency and OLS";
run;
```