



**POLITECHNIKA  
GDAŃSKA**

WYDZIAŁ FIZYKI TECHNICZNEJ  
I MATEMATYKI STOSOWANEJ

Imię i nazwisko studenta: Jakub Persjanow  
Nr albumu: 167766  
Studia pierwszego stopnia  
Forma studiów: stacjonarne  
Kierunek studiów: Fizyka Techniczna  
Specjalność: Informatyka stosowana

## PRACA DYPLOMOWA INŻYNIERSKA

Tytuł pracy w języku polskim: Badania obliczeniowe samoporzadkowania pryzmy piasku na wirującym dysku.

Tytuł pracy w języku angielskim: Computational study of self-organized phenomena of a sandpile on a rotating disc.

Potwierdzenie przyjęcia pracy	
Opiekun pracy  podpis	Kierownik Katedry/Zakładu (pozostawić właściwe)  podpis
dr hab. Jan Franz	

Data oddania pracy do dziekanatu:



**GDAŃSK UNIVERSITY  
OF TECHNOLOGY**


FACULTY OF APPLIED PHYSICS AND MATHEMATICS

Student's name and surname: Jakub Persjanow  
ID: 167766  
Undergraduate studies  
Mode of study: Full-time studies  
Field of study: Technical Physics  
Specialization: Applied Computer Science

## ENGINEERING DIPLOMA THESIS

Title of thesis: Computational study of self-organized phenomena of a sandpile on a rotating disc.

Title of thesis (in Polish): Badania obliczeniowe samoporzadkowania pryzmy piasku na wirującym dysku.

Supervisor	Head of Department
 signature	 signature
dr hab. Jan Franz	

Date of thesis submission to faculty office:



**GDAŃSK UNIVERSITY  
OF TECHNOLOGY**

FACULTY OF APPLIED PHYSICS AND MATHEMATICS

**DECLARATION regarding the diploma thesis titled:  
Computational study of self-organized phenomena of a sandpile on a  
rotating disc.**

First name and surname of student: Jakub Persjanow

Date and place of birth: 14.12.1997, Suwałki

ID: 167766

Faculty: Faculty of Applied Physics and Mathematics

Field of study: technical physics

Cycle of studies: undergraduate

Mode of study: Full-time studies

Aware of criminal liability for violations of the Act of 4<sup>th</sup> February 1994 on Copyright and Related Rights (Journal of Laws 2018, item 1191 with later amendments) and disciplinary actions set out in the Act of 20<sup>th</sup> July 2018 on the Law on Higher Education and Science (Journal of Laws 2018 item 1668 with later amendments),<sup>1</sup> as well as civil liability, I declare that the submitted diploma thesis is my own work.

This diploma thesis has never before been the basis of an official procedure associated with the awarding of a professional title.

All the information contained in the above diploma thesis which is derived from written and electronic sources is documented in a list of relevant literature in accordance with art. 34 of the Copyright and Related Rights Act.

I confirm that this diploma thesis is identical to the attached electronic version.

Gdańsk, .....

.....  
*signature of the student*

## **ABSTRACT**

The topic of this engineering thesis is a computational study of the self-organizing critical system phenomena. The main goal was to design and implement a two-dimensional model simulating a sandpile on a rotating disc and study whether it can be categorized as a self-organizing critical system. In the first chapter critical systems and their behavioral patterns are described. Then the sandpile model and self-organizing phenomena are described and self-organized system behavioral patterns are characterized. Also the motivation for writing the thesis is outlined in this chapter. The second chapter details the main goal of the thesis and the hypothesis is formulated. The third chapter describes the implementation of one and two-dimensional models. Data analysis and the comparison between the control group and the researched model is described, which is used for verifying the hypothesis. The fifth chapter contains the summary and the confirmation of the hypothesis.

**Keywords:** self-organization phenomenon, critical systems, computational study

## **STRESZCZENIE**

Poniższa praca opisuje badania obliczeniowe nad fenomenem samoorganizującego się systemu krytycznego. Głównym celem pracy było stworzenie i zaimplementowanie dwuwymiarowego modelu symulującego pryzmę piasku na wirującym dysku oraz przeanalizowanie czy dany system może zostać zakwalifikowany jako samoorganizujący się system krytyczny. W pierwszym rozdziale, wprowadzającym do omawianego zagadnienia, zostały opisane wzorce behawioralne systemów krytycznych oraz historia implementacji modelu komputerowego pryzmy piasku. Drugi rozdział wyszczególnia główne założenia pracy, została tam również zamieszczona sformułowana teza. Opis modeli wraz z opisem ich dynamiki zostały opisane w rozdziale trzecim. Rozdział czwarty poświęcony został na analizę danych oraz porównanie grupy kontrolnej i studiowanego stworzonego modelu wraz z jego pochodnych. Analiza miała na celu potwierdzenie hipotezy, co zostało opisane w rozdziale piątym wraz z podsumowaniem.

**Słowa kluczowe:** samoorganizacja, systemy krytyczne, badania obliczeniowe

## TABLE OF CONTENTS

LIST OF IMPORTANT SYMBOLS AND ABBREVIATIONS .....	7
1. INTRODUCTION .....	8
1.1 Critical systems and their behavioral patterns .....	8
1.2 Self – Organized Criticality .....	8
1.3 Sandpile Model .....	9
1.4 Motivation.....	9
2. MAIN GOAL OF THE THESIS.....	10
2.1 Hypothesis.....	10
3. MODELS DESCRIPTIONS .....	11
3.1 Overall description.....	11
3.2 One-dimensional model.....	11
3.2.1 Model description .....	11
3.2.2 Model dynamics.....	11
3.2.3 Boundary Conditions.....	12
3.3 Two-dimensional model with a quadratic grid .....	12
3.3.1 Model description .....	12
3.3.2 Model dynamics.....	12
3.3.3 Boundary Conditions.....	13
3.4 Two-dimensional model on a circular disc.....	13
3.4.1 Model description .....	13
3.4.2 Model dynamics.....	14
3.4.3 Boundary conditions.....	15
3.5 Two-dimensional model on a rotating circular disc.....	16
3.5.1 Differences between models on a circular disc .....	16
3.5.2 Simulated centrifugal and Coriolis forces .....	17
3.5.3 Value relocation with simulated rotational movement.....	17
4. DATA ANALYSIS .....	19
4.1 Main goal and description of the analysis .....	19
4.2 Data analysis of one-dimensional and two-dimensional model with quadratic grid .	19
4.2.1 One-dimensional model data analysis .....	19
4.2.2 Two-dimensional model with a quadratic grid data analysis .....	21
4.2.3 Two-dimensional model with a quadratic grid as a control group.....	24
4.3 Data analysis of the two-dimensional model on a circular disc.....	25
4.3.1 Data comparison with control group .....	27

4.4	Data analysis of the two-dimensional model on a rotating circular .....	29
4.4.1	Data comparison between two-dimensional on a circular disc and two-dimensional model on a rotating circular disc .....	32
4.4.1	Data comparison between control group and two-dimensional model on a rotating circular disc .....	32
4.4.2	Impact examination of changing the randomness of value addition in two-dimensional model on a rotating circular disc .....	33
5.	SUMMARY .....	35
5.1	Classification of designed model, hypothesis confirmation.....	35
	BIBLIOGRAPHY.....	36
6.	APPENDIX A: One-dimensional model python code .....	39
7.	APPENDIX B: Two-dimensional model with a quadratic grid python code.....	40
8.	APPENDIX C: Two-dimensional model with a circular rotating disc python code .....	42

## LIST OF IMPORTANT SYMBOLS AND ABBREVIATIONS

$i$	– current $x$ iteration position
$j$	– current $y$ iteration position
$N, M$	– maximum user defined numbers
$r_0$	– minimum integer radial value
$r_N$	– maximum integer radial value
$\vartheta_0$	– minimum integer angle value
$\vartheta_M$	– maximum integer angle value
$Z_{MAX}$	– maximum value amounts
SOC	– Self-organized criticality



## 1. INTRODUCTION

### ***1.1 Critical systems and their behavioral patterns***

To understand what is self-organized criticality one must first understand what are critical systems and their behavioral similarities to natural systems. In general they are distinguished by the outcome of system failure. Most of critical systems demonstrate similar behaviors and properties:

- When conditions are not sufficient critical systems are unstable, one divergence in the structure can cause destabilization that leads to disturbing phenomena, which affect multiple units of the system. Not only that, individual units affect neighboring units et cetera. Simply put, one small local change in one place impacts the whole system
- Occurrence of critical points. After reaching this point, the system strives to sustain stability in the critical state. Depending on examined model, they can be unit or environmental divergent.
- After reaching critical point, the system enters a critical state.

Seemingly critical systems are chaotic, but they can be described using mathematical apparatus.

### ***1.2 Self – Organized Criticality***

It can be observed that many natural systems exhibit behavioral patterns of critical systems. But if the critical points are chaotic, and consequently critical systems are chaotic, they should not be common in nature, as it is observed that natural systems strive to stability. This is the perplex that Bak, Tang and Wiesenfeld addressed in their studies [2], and came up with a solution they called self-organized criticality, SOC in short. „Self-organized” means that system proceeds from its original state towards a critical limit and suspends in that state without peripheral regulations. This definition cannot be treated as an ultimate definition, because there exists no general theory of self-organized critical system. Even though there is no definition it does not mean that there is no possibility to characterize a certain model as self-organizing. In order to achieve this following conditions must be met:

- the system must be open, values or any other variables added must have a possibility to leave the system;
- values or variables addition must be slow and forced;
- the system must be exposed to a local change that causes a critical instability that tries to be stabilized by some kind of readjustment.

### **1.3 Sandpile Model**

The sandpile model, proposed by Bak, Tang and Wiesenfeld in 1987 [2], is an abstract model representing a physical system with huge number of elements that go into interaction with their neighbors . It is a 2-D cellular automaton where the cell state represents the slope and height of a sandpile. The sandpile is checked every step if the critical point was reached, usually set to an integer number. If so the “topple” starts and relocates “sand”, which is a value corresponding to the state, to adjoined cells. When the “topple” arrives at the edge of the model, the excess “sand” is “spilled” over the edge, so only the value of the cell affected is decreased. Usually a single addition of “sand” does not cause much changes in the whole model, but when the whole system reaches the critical point the disturbance phenomena starts, called an “avalanche”. It affects a substantial part of the grid and when it is started the whole system sustains in a critical state, “avalanches” and “topples” are becoming more common. Seeing those results the authors concluded that the sandpile model displays self-organized criticality, it evolved from its basic state to a critical state and sustained in that state without outside interference.

### **1.4 Motivation**

The motivation for writing this thesis was to delve into and learn about very interesting critical systems and self-organizing phenomena. Critical systems can not only be used in a clear theoretical usage but also for a real world use. They can be used to simulate traffic, forest fires and evacuation procedures, that can help in estimating and predicting aforementioned situations. Also, the implemented model can be developed further into a more physically realistic system, to examine how the physical system behaves in comparison to cellular automaton.

## **2. MAIN GOAL OF THE THESIS**

The main goal of the thesis is to implement and study the self-organized phenomena of a sandpile model on a rotating disc, furthered called two-dimensional model on a rotating circular disc. To check whether the system, when changed from cartesian coordinates to polar coordinates and introduced to simulated movement, still behaves within the limits and whether it is possible to describe it using critical system descriptions. The work was divided into four main objectives:

- implementation of one-dimensional sandpile model and gathering data,
- implementation of two-dimensional cellular automaton of the sandpile model and gathering data,
- designing and implementing two-dimensional cellular automaton with polar coordinates and gathering data,
- designing two-dimensional cellular automaton with polar coordinated and introducing a simulated rotational movement and also gathering data.

Finally, the gathered data was analyzed and compared between all implemented models. Additionally, impact of changing the randomness of value addition in the final model was examined.

### **2.1 Hypothesis**

Does the implemented two-dimensional model on a rotating circular disc exhibits behavioral patterns of a self-organizing critical system and can it be characterized as one?

### 3. MODELS DESCRIPTIONS

#### 3.1 Overall description

As mentioned in the main goal of the thesis subchapter, models were programmed using the Python programming language in version 3.7 [11]. For the visual representation and graph generation the library matplotlib [9] was used and for data gathering and generation the numpy library [10].

#### 3.2 One-dimensional model

##### 3.2.1 Model description

The one-dimensional model was solely based on a model described in the book "Natural Complexity" by Paul Charbonneau [1]. It is a one-dimensional cellular automaton consisting of  $N$  cells with only two neighboring cells, on the right and left. The whole range of the model is defined by the equation:

$$S_j^0 = 0, \quad j = 0, \dots, N - 1$$

Where the variable  $S_j^n$  describes the latitude,  $j$  identifies the cell on the given lattice and  $n$  indicates the number of the iteration. During every iteration, a value of  $s$  is added to the  $S$  variable to a pseudo-randomly selected node, defined by equation:

$$S_r^{n+1} = S_r^n + s, \quad r \in [0, N - 1], \quad s \in [0, \varepsilon]$$

Where  $\varepsilon$  is the maximum increment defined by the user.

The physical equivalent that inspired the model is a sandpile, so essentially  $S_j^n$  is the height measurement of a pile with the position of  $j$  and at the time of  $n$ . As it can be expected the "sandpile" will grow in time with values defined by  $s$ .

##### 3.2.2 Model dynamics

As the growth of a pile is progressing, at every iteration the whole "pile" is checked and the slope is calculated using the following equation:

$$z_j^n = |S_{j+1}^n - S_j^n|, \quad j = 0, \dots, N - 2$$

If the slope exceeds the critical amount  $Z_c$ , which is defined by the user, the pair of two cells ( $j$ ,  $j + 1$ ) is deemed unstable. It is a critical state that leads to a "topple". It is a redistribution process that displaces a certain quantity of values from the cell with the higher  $S_j^n$  to the neighboring cells. Redistribution process is used to restore local stability.

### 3.2.3 Boundary Conditions

At the last cell at every iteration any accumulated value is being removed due to a redistribution process.

$$S_{N-1}^n = 0$$

For data analysis purposes “removed” values are stored in an array.

## 3.3 Two-dimensional model with a quadratic grid

### 3.3.1 Model description

The two-dimensional model is a cellular automaton, consisting of  $N \times M$  number of cells, each described by  $x$  and  $y$  coordinates with four neighboring cells. The implementation was based on [3 – 7]. The whole range, a grid, of the model can be described using a two-dimensional matrix:

$$\begin{bmatrix} Z(x_0, y_0) & \cdots & Z(x_N, y_0) \\ \vdots & \ddots & \vdots \\ Z(x_0, y_M) & \cdots & S(x_N, y_M) \end{bmatrix}$$

where:

$N, M$  – maximum user defined even numbers

In every iteration a small value of  $s$  is added to the central point of the model defined by the equation:

$$Z\left(\frac{x_N}{2}, \frac{y_M}{2}\right) + s, \quad s = 1$$

Physical inspiration, as in the one-dimensional model, is a sandpile, where the center point  $Z\left(\frac{x_N}{2}, \frac{y_M}{2}\right)$  is the peak of the sandpile and it grows in time with values defined by  $s$ .

### 3.3.2 Model dynamics

At every iteration, all cells are checked whether the maximum limit of value ( $Z_{MAX}$ ) able to be accumulated by each cell is exceeded. When it is, the redistribution process, “topple”, is launched and all accumulated values are displaced to four neighboring cells. The redistribution is done by the following rules:

if  $Z(x_i, y_j) > Z_{MAX}$  then:

$$\begin{aligned} Z(x_i, y_j) &\rightarrow Z(x_i, y_j) - 4 \\ Z(x_{i+1}, y_j) &\rightarrow Z(x_{i+1}, y_j) + 1 \\ Z(x_{i-1}, y_j) &\rightarrow Z(x_{i-1}, y_j) + 1 \\ Z(x_i, y_{j+1}) &\rightarrow Z(x_i, y_{j+1}) + 1 \\ Z(x_i, y_{j-1}) &\rightarrow Z(x_i, y_{j-1}) + 1 \end{aligned}$$

where:

$i$  – current  $x$  iteration position  
 $j$  – current  $y$  iteration position

### 3.3.3 Boundary Conditions

When the redistribution process is started in a cell, that is close to the border of the grid the excess values are removed. Described by following equation:

if  $i \leq 0$  or  $i \geq N$  or  $j \leq 0$  or  $j \geq M$  then:

$$Z(x_i, y_j) \rightarrow 0$$

For data analysis purposes “removed” values are stored in an array.

## 3.4 Two-dimensional model on a circular disc

### 3.4.1 Model description

In the original sandpile model the grid was two-dimensional and the location of each cell was described using cartesian coordinates. For the purposes of the thesis, as the final model is supposed to simulate rotational movement, polar coordinates were implemented instead. The whole grid, from the programming point of view, is a two-dimensional array,  $\vartheta$  and  $r$  were used to describe each cell location on the grid. The grid can be described using following matrix:

$$\begin{bmatrix} Z(r_0, \vartheta_0) & \cdots & Z(r_N, \vartheta_0) \\ \vdots & \ddots & \vdots \\ Z(r_0, \vartheta_M) & \cdots & Z(r_N, \vartheta_M) \end{bmatrix}$$

where:

$N, M$  – maximum user defined even numbers

$r_0$  – minimum integer radial value

$\vartheta_0$  – minimum integer angle value

$r_N$  – maximum integer radial value

$\vartheta_M$  – maximum integer angle value

#### 3.4.1.1 Graphical Representation

For graphical representation, conversion from linear arrays using *numpy* library method *linspace* was done. Following methods represent the conversions:

$$\text{numpy.linspace}(0, 2 * \pi, M + 1)$$

$$\text{numpy.linspace}(0, r_i, N + 1)$$

where:

$M, N$  – maximum user defined integer value

$i$  – current x iteration position

Transformations were done for the *matplotlib* library to associate a two-dimensional array with a graphical representation that uses polar coordinates.

#### 3.4.1.1.1 Graph

Using the aforementioned transformations, the *matplotlib* library is used to project the two-dimensional array as a plot with polar coordinates.

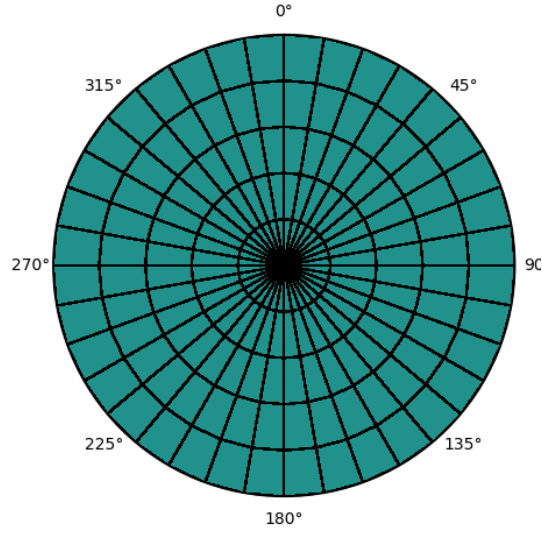


fig. 3.1 visual representation of two-dimensional model on a circular disc

Figure 3.1 represents two-dimensional array using polar coordinates, each visible cell holds values generated during simulation.

#### 3.4.2 Model dynamics

##### 3.4.2.1 Value addition

At each iteration a value of 1 is added to the system in consistent  $r_0$  and pseudo-random  $\vartheta_j$  location, defined by the following equation:

$$Z(r_0, \vartheta_j) = Z(r_0, \vartheta_j) + 1, \quad j \in [0, M]$$

where:

- $M$  – maximum user defined integer value
- $r_0$  – minimum integer radial value
- $j$  – integer angle value at current iteration

### 3.4.2.2 Value relocation

At every iteration, all cells are checked whether the maximum limit of value  $Z_{MAX}$  able to be accumulated by each cell is exceeded. When it is, the redistribution process, “topple”, is launched and some quantity of values is displaced to three neighboring cells

If  $Z(r_i, \vartheta_j) > Z_{MAX}$  then:

$$Z(r_i, \vartheta_j) \rightarrow Z(r_i, \vartheta_j) - 3$$

$$Z(r_{i+1}, \vartheta_j) \rightarrow Z(r_{i+1}, \vartheta_j) + 1$$

$$Z(r_i, \vartheta_{j+1}) \rightarrow Z(r_i, \vartheta_{j+1}) + 1$$

$$Z(r_i, \vartheta_{j-1}) \rightarrow Z(r_i, \vartheta_{j-1}) + 1$$

where:

$i$  – current x iteration position

$j$  – current y iteration position

In the previous models the values were relocated to four neighboring cells but, as the real life equivalent, values should not be displaced “upwards”.

### 3.4.3 Boundary conditions

As the described model is a circular grid with polar coordinates, boundary conditions only occur at the edges where the diameter value  $r_N$  is exceeded. Excess values are removed from the grid and are stored for later analysis. The relocation at the boundary relocation can be describe with the following equations

if  $r_i = r_N$  then:

$$Z(r_i, \vartheta_j) \rightarrow Z(r_i, \vartheta_j) - 3$$

the  $Z(r_{i+1}, \mu_j) \rightarrow Z(r_{i+1}, \mu_j) + 1$  relocation does not occur but

$$Z(r_i, \vartheta_{j+1}) \rightarrow Z(r_i, \vartheta_{j+1}) + 1$$

$$Z(r_i, \vartheta_{j-1}) \rightarrow Z(r_i, \vartheta_{j-1}) + 1$$

When the value of  $\vartheta$  is exceed,  $\vartheta_i > \vartheta_M$ , the values are relocated to the neighboring cells (as if they were on the circle plot) using the following equations:

first condition:

if  $\vartheta_j = \vartheta_M$  then:

$$Z(r_i, \vartheta_j) \rightarrow Z(r_i, \vartheta_j) - 3$$

$$Z(r_{i+1}, \vartheta_j) \rightarrow Z(r_{i+1}, \vartheta_j) + 1$$

$$Z(r_i, \vartheta_0) \rightarrow Z(r_i, \vartheta_0) + 1$$

$$Z(r_i, \vartheta_{j-1}) \rightarrow Z(r_i, \vartheta_{j-1}) + 1$$



second condition:

if  $\vartheta_j = \vartheta_0$  then:

$$Z(r_i, \vartheta_j) \rightarrow Z(r_i, \vartheta_j) - 3$$

$$Z(r_{i+1}, \vartheta_j) \rightarrow Z(r_{i+1}, \vartheta_j) + 1$$

$$Z(r_i, \vartheta_{j+1}) \rightarrow Z(r_i, \vartheta_{j+1}) + 1$$

$$Z(r_i, \vartheta_M) \rightarrow Z(r_i, \vartheta_M) + 1$$

where:

$N, M$  – maximum user defined integer value

$i$  – current x iteration position

$j$  – current y iteration position

### 3.5 Two-dimensional model on a rotating circular disc

#### 3.5.1 Differences between models on a circular disc

Comparing last model with the final one, the main foundations and description does not differ as much. The model still can be described using following matrix

$$\begin{bmatrix} Z(r_0, \vartheta_0) & \cdots & Z(r_N, \vartheta_0) \\ \vdots & \ddots & \vdots \\ Z(r_0, \vartheta_M) & \cdots & Z(r_N, \vartheta_M) \end{bmatrix}$$

where:

$N, M$  – maximum user defined integer value

$r_0$  – minimum integer radial value

$\vartheta_0$  – minimum integer angle value

$r_N$  – maximum integer radial value

$\vartheta_M$  – maximum integer angle value

Two additional arrays generated using the *numpy* library are still used and graphical representation is exactly the same. Also, the value addition applied in this model are equivalent to the two-dimensional model without simulated rotational movement. Only differences are in value relocation as a simulated centrifugal force is implemented, the “disc” is “rotating” as described below.

### 3.5.2 Simulated centrifugal and Coriolis forces

Even though the main model is supposed to simulate centrifugal and Coriolis forces and it is based on a physical model, the whole “force” is an approximated influence, which aforementioned forces have on a small physical objects. Values gathered in each cell have no mass, they do not interact with each other, or with neighboring cells, in physical meaning. No friction or gravitational forces are considered. They are “moved” in a certain way though while value relocation is taking place.

### 3.5.3 Value relocation with simulated rotational movement

As before, at every iteration when the maximum amount of values is gathered in each cell the “topple”, value relocation, occurs and can be described using the following equations

if  $Z(r_i, \vartheta_j) > Z_{MAX}$  then:

$$Z(r_i, \vartheta_i) \rightarrow Z(r_i, \vartheta_j) - 3$$

$$Z(r_{i+1}, \vartheta_j) \rightarrow Z(r_{i+1}, \vartheta_j) + 1$$

$$Z(r_i, \vartheta_{j+1}) \rightarrow Z(r, \vartheta_{j+1}) + 1$$

$$Z(r_{i+1}, \vartheta_{j+1}) \rightarrow Z(r_{i+1}, \vartheta_{j+1}) + 1$$

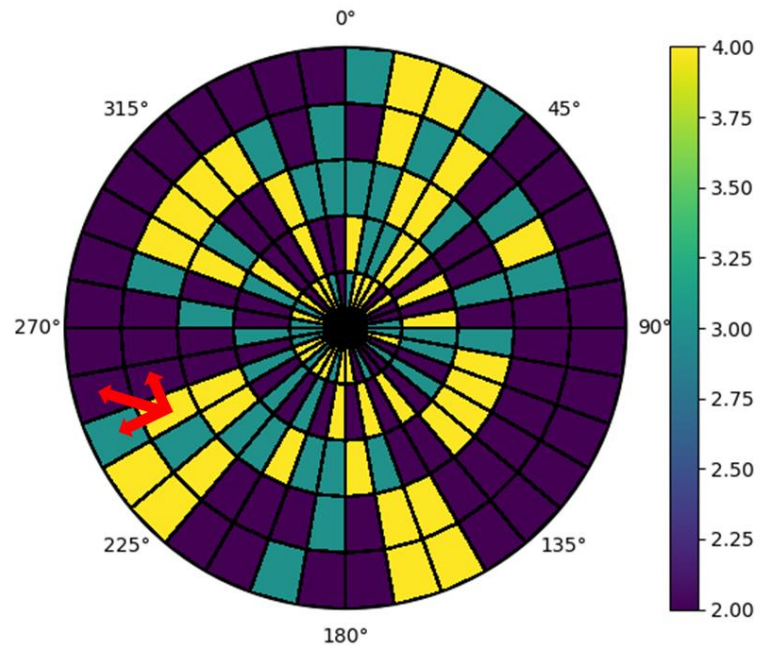
where:

$Z_{MAX}$  – maximum value amounts

$i$  – current x iteration position

$j$  – current y iteration position

While the simulated forces are applied, the values are relocated into two neighboring cells and one cell that is cross-wise from the primary one where “topple” is occurring. Visualized in fig. 3.2.



*fig. 3.2 visual representation of value relocation marked as red arrows starting from yellow cell*

## **4. DATA ANALYSIS**

### ***4.1 Main goal and description of the analysis***

The main goal of the analysis is to check, whether the designed model exhibits critical self-organizing system behaviors. By comparison of the one-dimensional and two-dimensional model with quadratic grid data to two-dimensional model on a rotating circular disc and by knowing the critical system behavioral patterns it is possible to assume that the implemented model indeed is a self-organizing critical system.

The analysis is divided into four main steps:

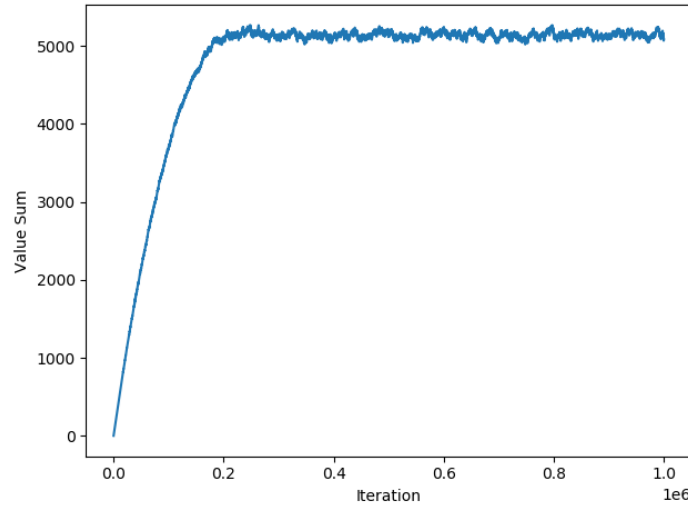
- Data analysis gathered from two-dimensional model.
- Comparison between the control group, which consists of the two-dimensional model with quadratic grid, to a two-dimensional model on a circular disc.
- Comparison between two-dimensional model on a circular disc and two-dimensional model on a rotating circular disc.
- Comparison between the control group and a two-dimensional model on a rotating circular disc.

Additionally, the random addition of values was implemented and analyzed to check, whether it exhibits any divergence from other implementations.

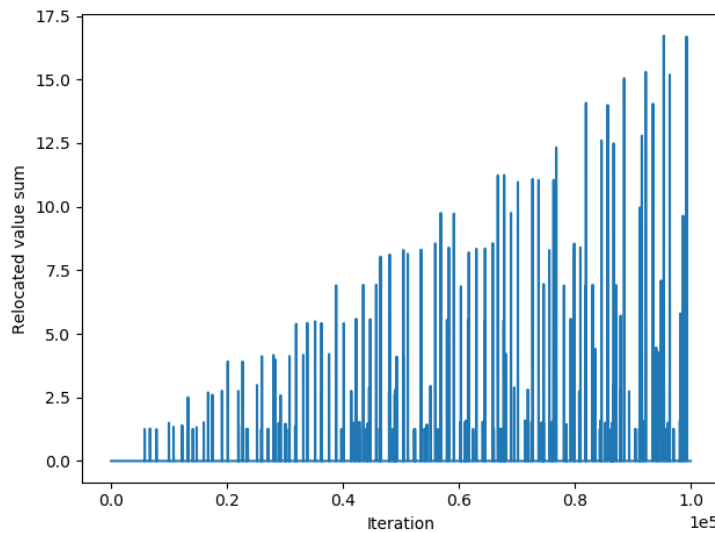
### ***4.2 Data analysis of one-dimensional and two-dimensional model with quadratic grid***

#### ***4.2.1 One-dimensional model data analysis***

During each iteration information about the sum of the values in the system and displaced values on the grid was gathered. In other words, sum of the values represent the total values gathered in the system and displaced values represent how many values were moved during relocation. In the physical system it would be equivalent to the mass of the sandpile and how much sand was displaced. Based on the data, two main plots were charted. One for 1'000'000 iterations the second for 100'000 iterations. The difference between iterations was done for clearer representation.

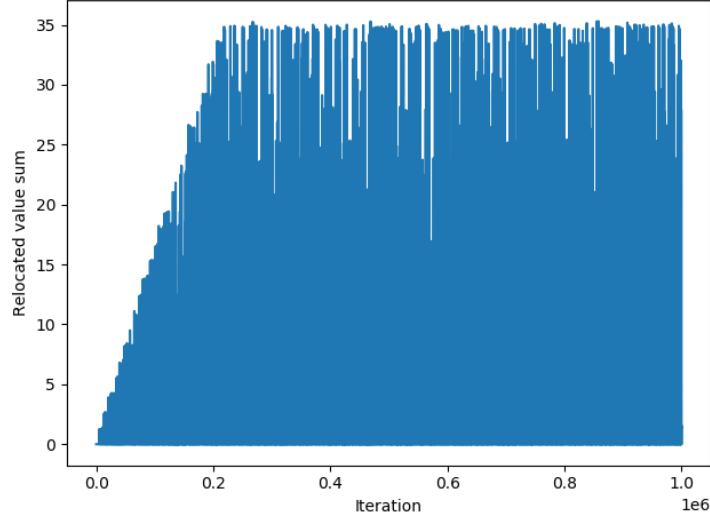


*fig. 4.1 time series of value sum accumulated in the one-dimensional system*



*fig. 4.2 time series of relocated values in the one-dimensional system*

As it can be observed on fig. 4.1, the system is accumulating values till around 200'000 iterations when dislocations started to occur. The self-organized definition states: the system moves from a stable state and sustains in that state for the rest of the iterations, this observation shows that the implemented one-dimensional model can, in approximation, be classified as self-organized critical system by that definition. By that we can conclude that at around 200'000 iteration system failure, which is a moment when the system entered a critical state, occurred. Fig. 4.2 further proves that statement. Relocated values occur more frequently and when occurring, the system reverts back to the critical state. As the iterations continue, the amounts of dislocated values increase but the system still strives to its critical state. It is better represented on fig. 4.2.1 with increased amount of iterations.

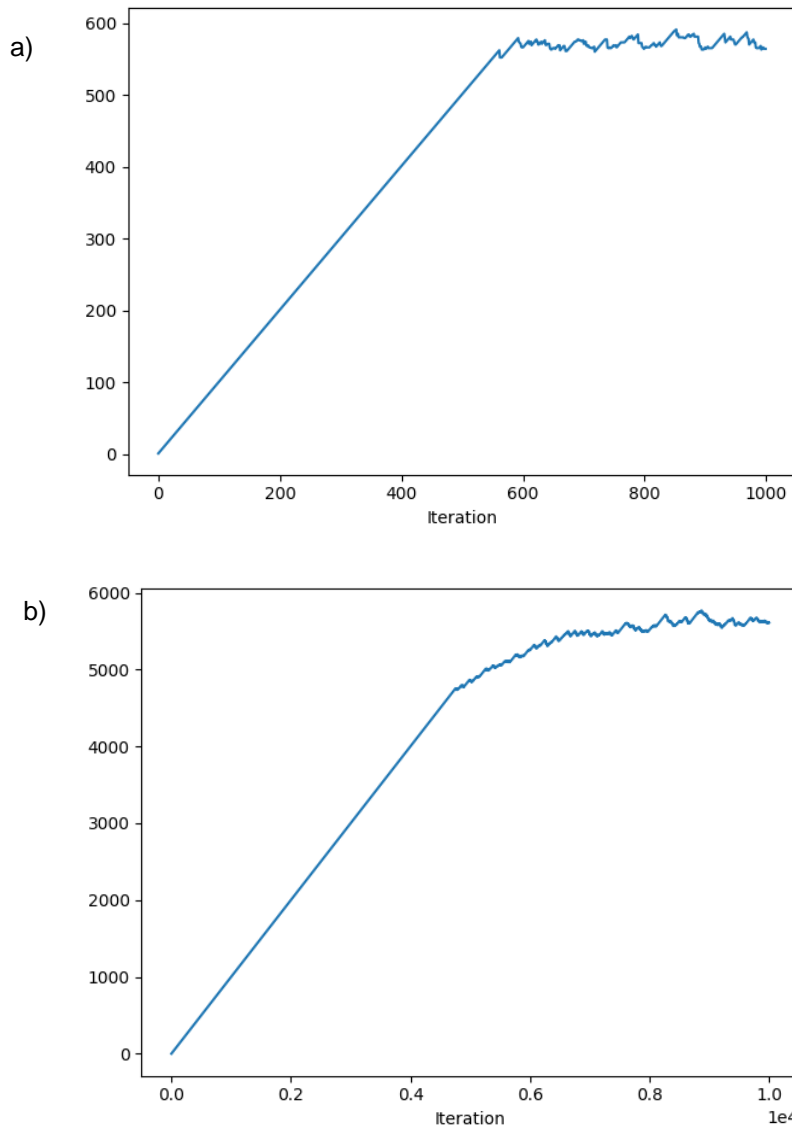


*fig. 4.2.1 time series of relocated values in the one-dimensional system for 1'000'000 iterations*

#### *4.2.2 Two-dimensional model with a quadratic grid data analysis*

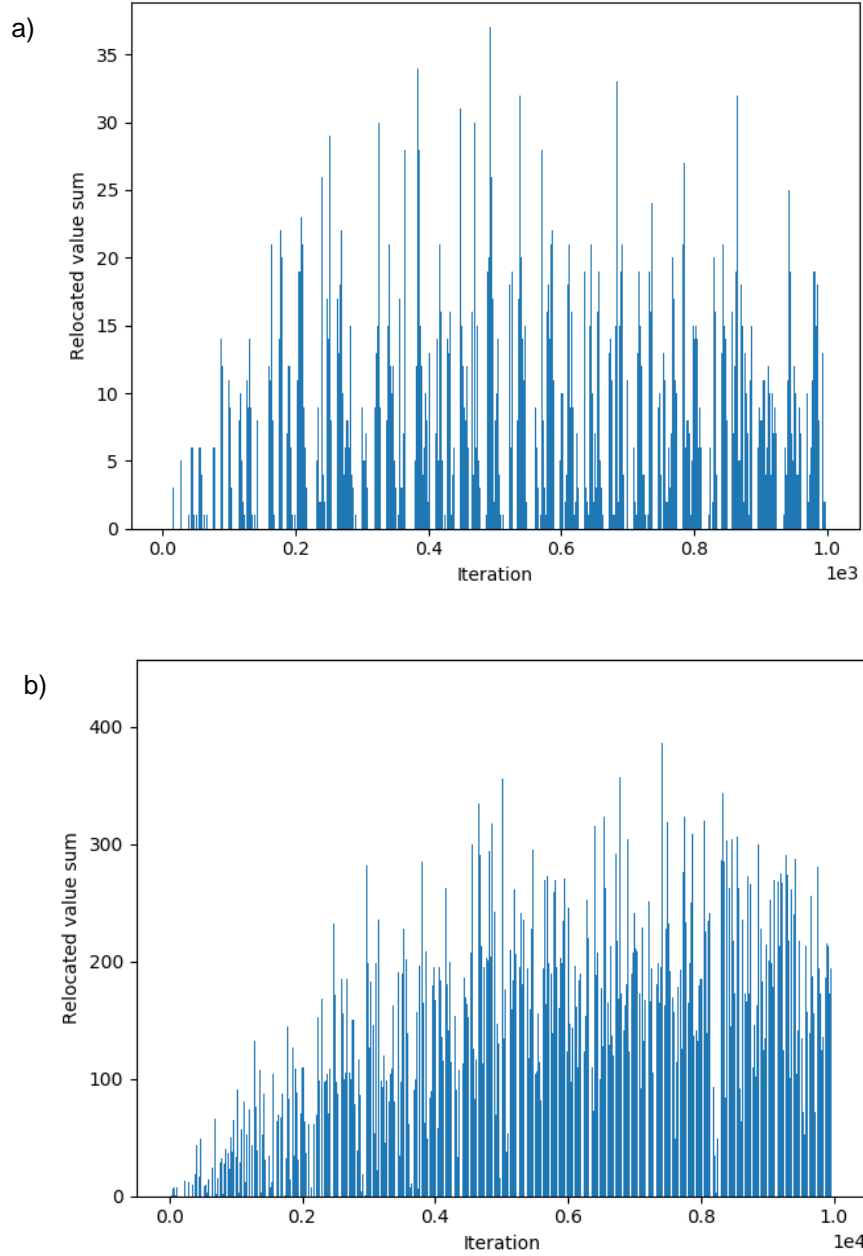
Similar as for one-dimensional model, during each repetition, the sum of the values and displaced values were gathered. Same plots were charted and the obtained results were expected although differ slightly from previous model. Simulation was run on a 11 by 11 grid, for 1000 iterations. Then on a 51 by 51 grid for 10'000 iterations to check if the system exhibits any variations. The maximum value amount that can be allocated in each cell was set to  $Z_{MAX} = 4$ , when this amount was exceeded the relocation occurred.

The simulations have been performed with two grid sizes to show that the same phenomena can be observed independent of the grid size. For a larger grid size, a larger number of iterations is required to observe similar patterns, which results in an increase in the computational time. In order to reduce the computer time for the further studies, more attention is paid on the system with smaller grid size.



*fig. 4.3 time series of value sum accumulated in the two-dimensional system  
a) for 1000 iteration b) for 10'000 iterations*

In both figures it can be observed that, when entering into its critical state, system persists in this state and oscillates around some values. The main difference is the point where the system enters the critical state. In fig. 4.3a it starts around 500<sup>th</sup> iteration and the oscillation starts almost immediately. In fig. 4.3b however, the critical state is achieved around 4000<sup>th</sup> iteration, then the oscillation starts slightly, but the values are still growing till around 7000<sup>th</sup> iteration. This is due to the grid size. Both outcomes are expected and the system behaves within the limits of the self-organizing criticality.



*fig. 4.4 time series of relocated values in the two-dimensional system  
a) for 1000 iteration b) for 10'000 iterations*

Fig. 4.4a and 4.4b describe the sum of the relocated value for each iteration. Similar to the one-dimensional model, for both grid sizes. The sum of the relocated values is most frequent around 10 for the smaller grid and around 150 for the larger grid. Interestingly enough examining fig. 4.4a, times when no relocations occur are visible around every 500 iterations.



#### 4.2.2.1 Graphical representation of two-dimensional model with a quadratic grid

Graphically the two-dimensional model was represented on a 51 by 51 grid with 10'000 iterations.

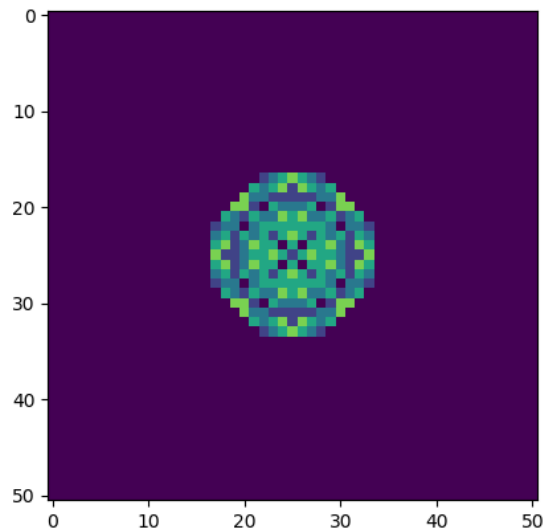


fig. 4.5 graphical representation of two-dimensional model with a quadratic grid after 1000 iterations

On fig. 4.5 it can be observed that visible patterns started to form on a grid. Different colored pixels represent different amounts of values gathered in one cell.

#### 4.2.3 Two-dimensional model with a quadratic grid as a control group

The aforementioned model, due to its categorization as self-organizing critical system, will be used as a control group for further analysis. If the two-dimensional model on a rotating circular disc will display behaviors similar to control group it is possible to assume that the designed implementation is indeed a self-organizing critical system.

#### 4.3 Data analysis of the two-dimensional model on a circular disc

Corresponding to control group, data was gathered from two-dimensional model on a circular disc: sum of the values, relocated values and also each cell value for graphical representation at respectively 1000 and 10'000 iteration. The grid consisted of  $M = 36$  angular grid points and  $N = 5$  radial grid points, totaling to 180 cells. Maximum value was set to  $Z_{MAX} = 3$ .

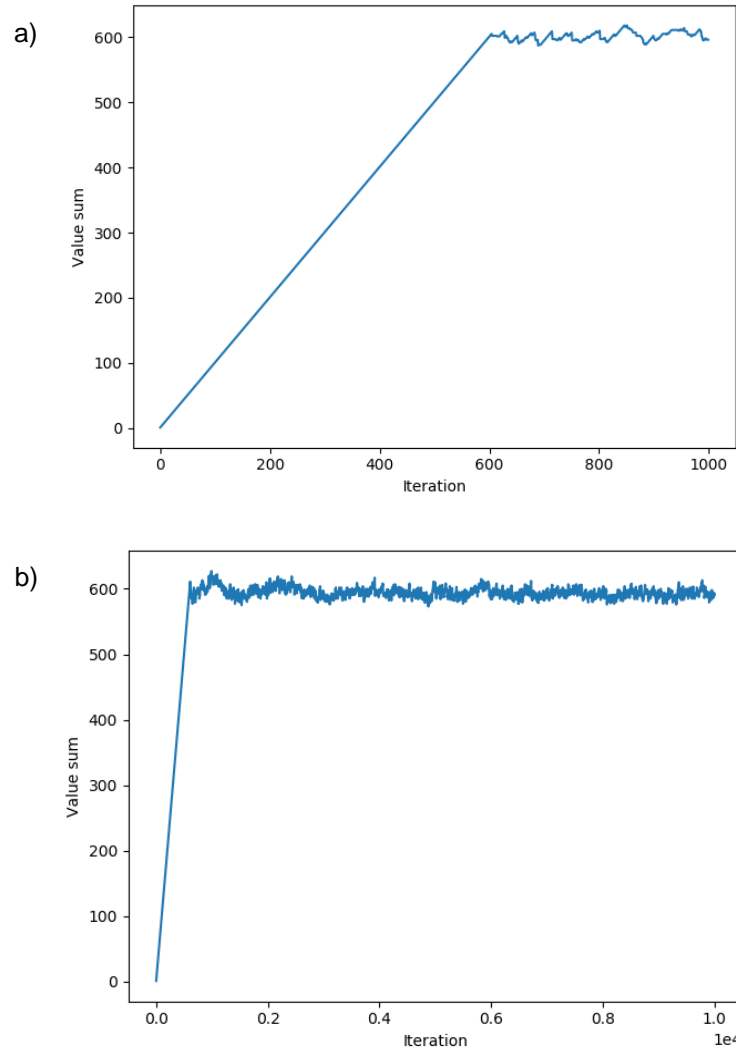
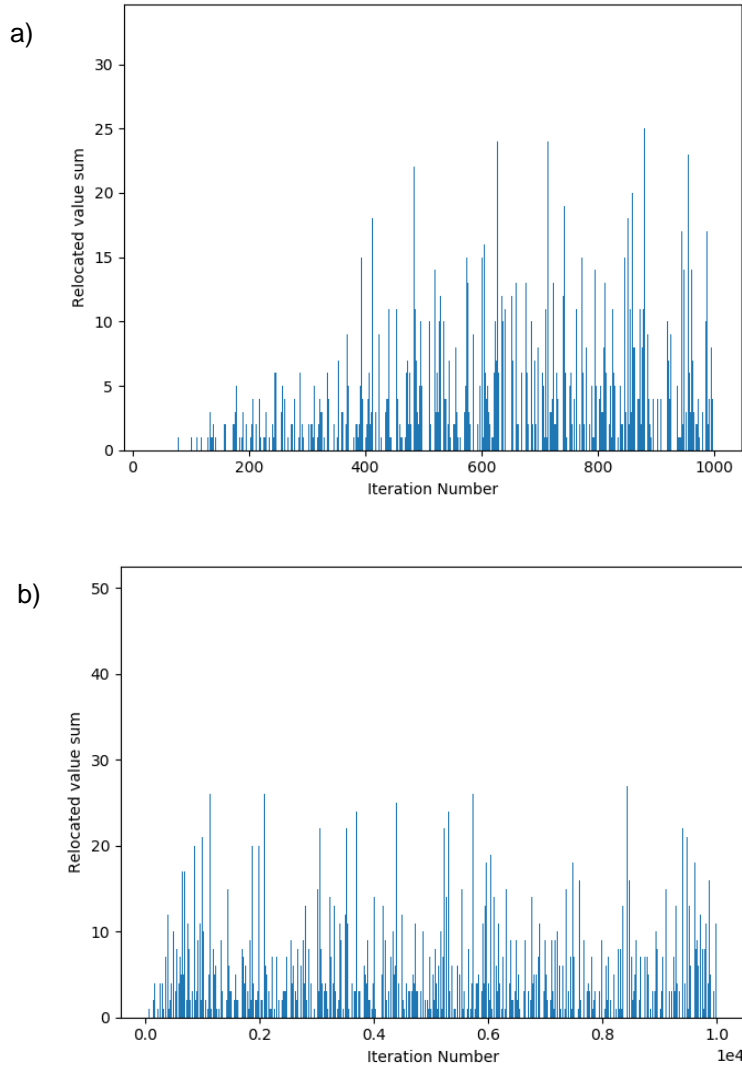


fig. 4.6 time series of value sum accumulated in the two-dimensional model on a circular disc  
a) for 1000 iterations b) for 10'000 iterations

Similarly to fig. 4.3a and fig. 4.3b the obtained results clearly indicate that the system entered a critical state, around 600<sup>th</sup> iteration, on both figures. During that state the oscillation around a constant value occurred and the system maintained in that state. The only differences are, again, the extent of the oscillation which is much more prominent in fig. 4.6b due to the significant increase in number of iterations.



*fig. 4.7 time series of relocated values in the two-dimensional system on a rotating circular disc  
a) for 1000 iterations b) for 10'000 iterations*

Fig. 4.7a and 4.7b represent the sum of the relocated values in each iteration, respectively for 1000 and 10'000 iterations. The maximum relocation amounts are fluctuating around 25 in both cases, which indicated that the designed model constrains stability. Also, the highest concentration of the relocation remains between:

- 0 and 5 for 1000 iterations,
- 0 and 10 for 10'000 iterations,

which further proves that the system, after entering the critical state, strives to stability.

#### 4.3.1 Data comparison with control group

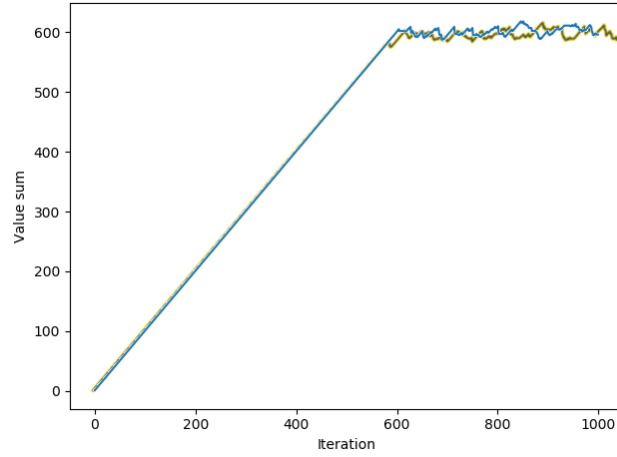


fig. 4.8 comparison between value sum gathered from two-dimensional control group model (yellow) and two-dimensional model on a circular disc (blue) for 1000 iterations

Fig. 4.8 presents the comparison of value sum to iteration number for 1000 iterations between two-dimensional model with quadratic grid from control group and two-dimensional model on a circular disc. Although the system failures occurs prior in the control model, both systems oscillate around the same constant value while trying to maintain stability. On this basis it can be concluded that the two-dimensional modal on a circular disc meets one of the requirements to be classified as a self-organizing critical system.

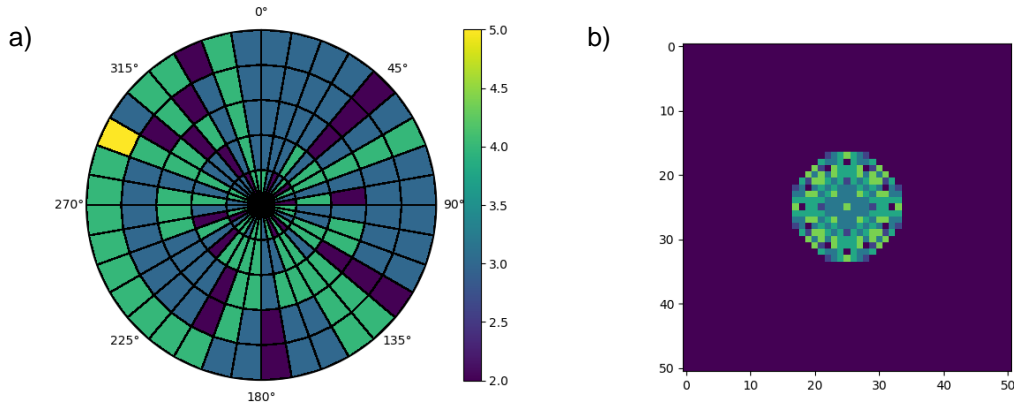
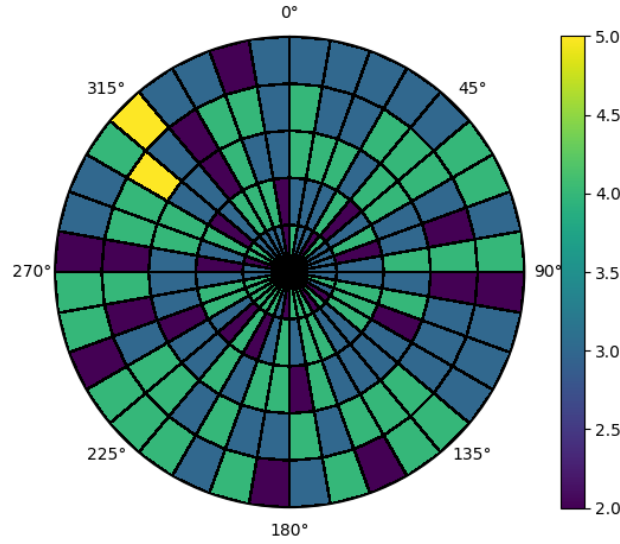


fig. 4.9 visual representation of  
a) two-dimensional model on a circular disc during 1000<sup>th</sup> iteration, each cell color coded with value amount currently occupying the cell  
b) control two-dimensional model with quadratic grid during 1000<sup>th</sup> iteration

Fig. 4.9a presents the model state during the 1000<sup>th</sup> iteration. Comparing it to fig. 4.9b, the two-dimensional model on a circular disc does not display any visible patterns by contrast to the control model, where patterns are clearly distinguishable. It is due to the value addition that differs between two models. In the control model the value addition is constant and takes places in  $Z(x_{\frac{N}{2}}, y_{\frac{M}{2}})$ , where as in two-dimensional model on a circular disc the middle point of the grid

does not exist (marked as black area in the middle of the chart). The values are added to cells with pseudo-random value of  $\vartheta_j$  and constant  $r_i = 1$ . Although the difference is significant it can be excluded as a deciding factor as the limitations of the *matplotlib* library and the technique of data plotting are a main cause.

Similar visualization was done after 10'000 iterations

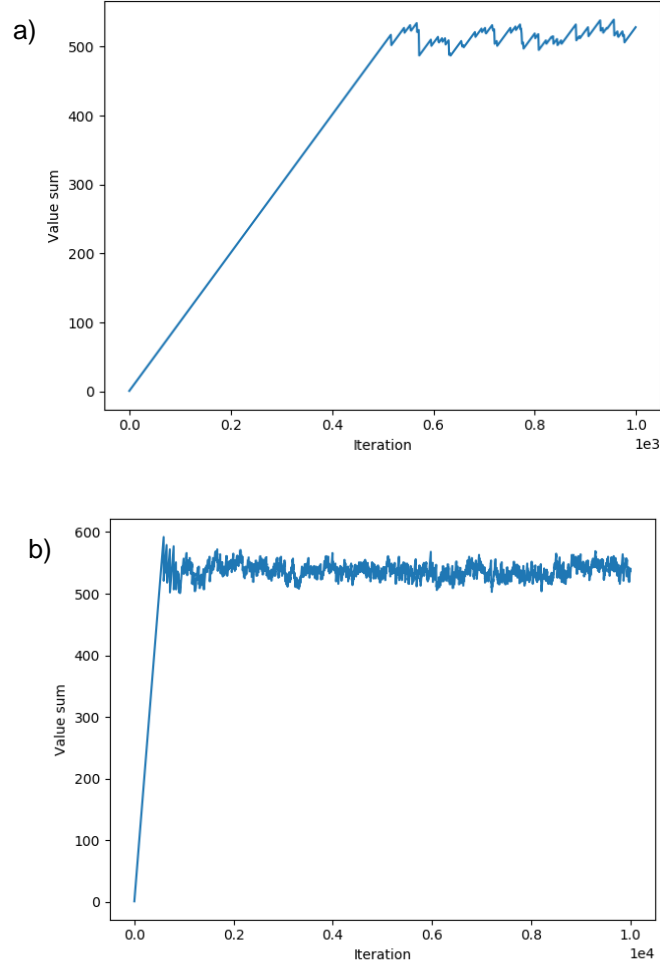


*fig. 4.10 visual representation of two-dimensional model on a circular disc during 10000<sup>th</sup> iteration*

As in fig. 4.9a, fig. 4.10 presents no classifiable or visible patterns.

#### 4.4 Data analysis of the two-dimensional model on a rotating circular

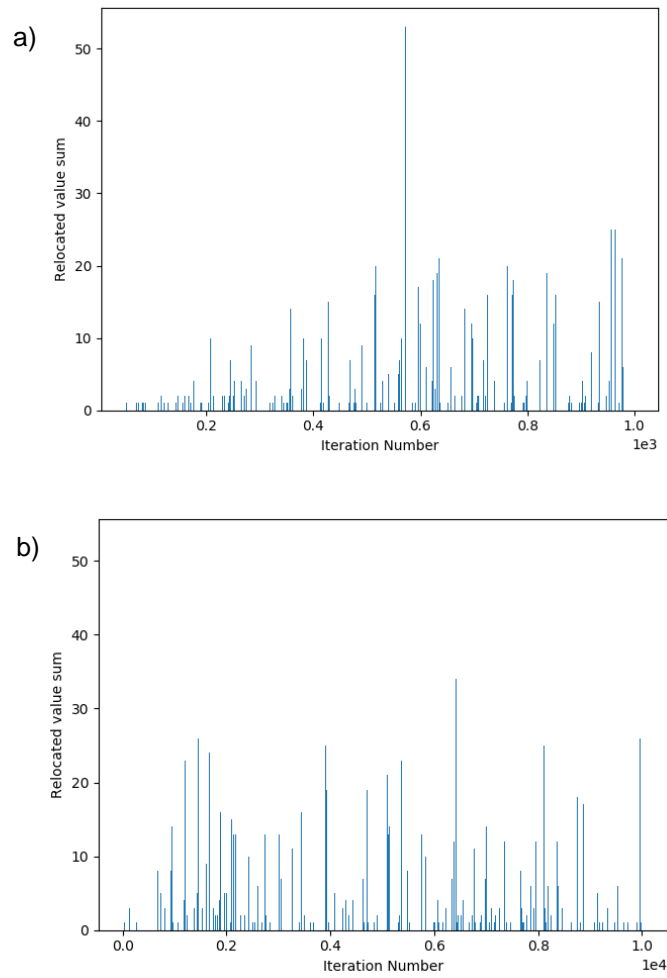
Equivalently to control group data from two-dimensional model on a rotating circular disc was gathered. Similarly as for model with without rotational movement, the sum of the values, relocated values and also each cell value for graphical representation at respectively 1000 and 10000 iteration were plotted. The grid consisted of  $M = 36$  angular grid points and  $N = 5$  radial grid points, totaling to 180 cells. Maximum value was set to  $Z_{MAX} = 3$ .



*fig. 4.11 time series of value sum accumulated in the two-dimensional model  
on a rotating circular disc  
a) for 1000 iterations b) for 10'000 iterations*

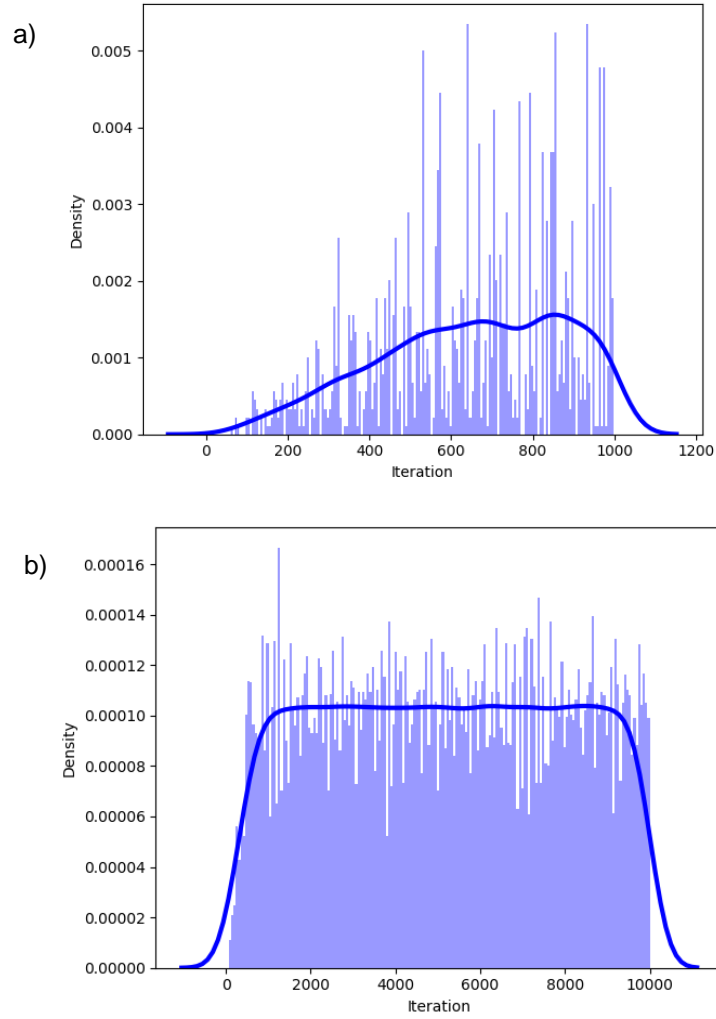
Fig. 4.11a represents the value sum of the system to iterations for 1000 repetitions. The magnitude of oscillation in comparison to two-dimensional model on a circular disc, represented on fig. 4.6a, is more prominent. While the system entered a critical state around 500<sup>th</sup> iteration, the value sum still progressed to accumulate till around 600<sup>th</sup> iteration. It is considerably harder to point a constant value around which the oscillation occurs. Same observation can be pointed on fig. 4.11b, which represents a simulation for 10'000 iterations. The system entered critical state around 500<sup>th</sup> iteration, the same as for 1000 iterations. The oscillation also shows higher

magnitude of divergence in comparison to fig. 4.6b. This is caused by the introduction of the simulated rotational movement.



*fig. 4.12 time series of relocated values in the two-dimensional system  
on a rotating circular disc  
a) for 1000 iterations b) for 10'000 iterations*

Fig. 4.12a and 4.12b represent the relocated value sum during each iteration. The maximum relocation amounts oscillate around 30, higher than in the model without rotational movement, represented in fig. 4.7a and 4.7b. Also the relocations occur less frequently. Even though the relocations are less frequent, the amounts of relocated values are much higher.



*fig. 4.13 density to iteration trend line (dark blue line), amounts of relocations during each iteration (light blue histogram)  
a) for 1000 iterations b) for 10'000 iterations*

Additionally, the density function was plotted for the simulation. Presented on fig. 4.13a and 4.13b, approximated amounts of each relocation for 1000 and 10'000 iterations accordingly. Trend line visible on fig. 4.13a presents that the relocation occurs more frequently as the repetition progresses. It indicates that the critical state has been achieved. Fig. 4.13b plotted for 10'000 iterations show different behavior. As the iteration amount is increased the trend line grows for first 1000 repetitions, then stabilizes forming an almost constant function. This indication proves that after reaching the critical state the system stabilizes and continues in that state.



#### 4.4.1 Data comparison between two-dimensional on a circular disc and two-dimensional model on a rotating circular disc

Data from figs. 4.11a, 4.11b, 4.12a and 4.12b indicate that, when rotation is introduced, the critical point of the system is moved, stability is lower than without the rotational movement. It is visualized in fig. 4.14

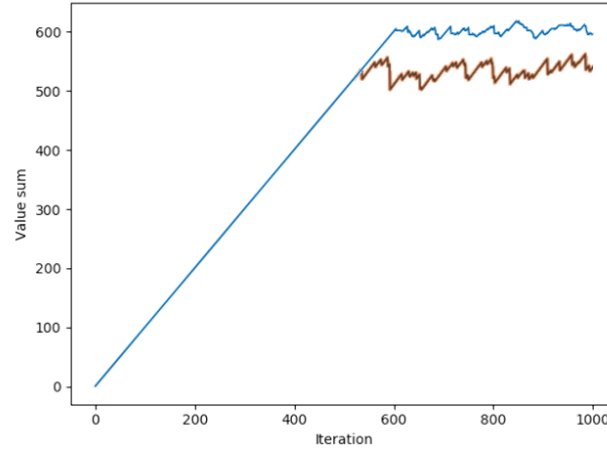


fig 4.14 comparison between two-dimensional model on a circular disc (red) and two-dimensional model on a rotating disc (blue) for 1000 iterations

It is clearly visible that the critical point was moved from around 600 to around 500 when simulated rotational movement was introduced. Also, the difference between the peaks of the maximum and minimum value sum during critical state are greater. It still indicates that the system can be classified as self-organized criticality. When it enters the critical state it still behaves as expected, trying to maintain stability.

#### 4.4.1 Data comparison between control group and two-dimensional model on a rotating circular disc

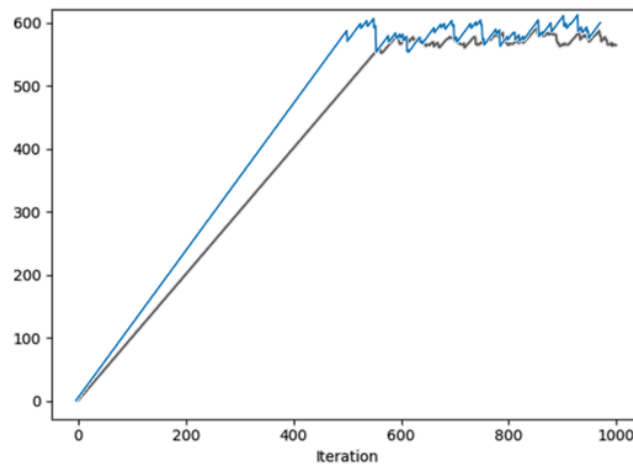


fig. 4.15 comparison between control group (grey) and two-dimensional model on a rotating circular disc (blue) for 1000 iterations

Fig. 4.15 represents the comparison of the value sum to iteration between control group and the two-dimensional model on a rotating circular disc. Just as in fig. 4.14, the initial growth of the values is more rapid. Consequently the entrance into the critical state occurs quicker and the oscillation is more prominent. The constant that the oscillation occurs around is also moved, from around 400 to around 450. The behavior that the figure indicates is expected, the system when entering the critical point continues with maintaining stability which is a clear indication of self-organized criticality, even if the differences between peak values are magnified.

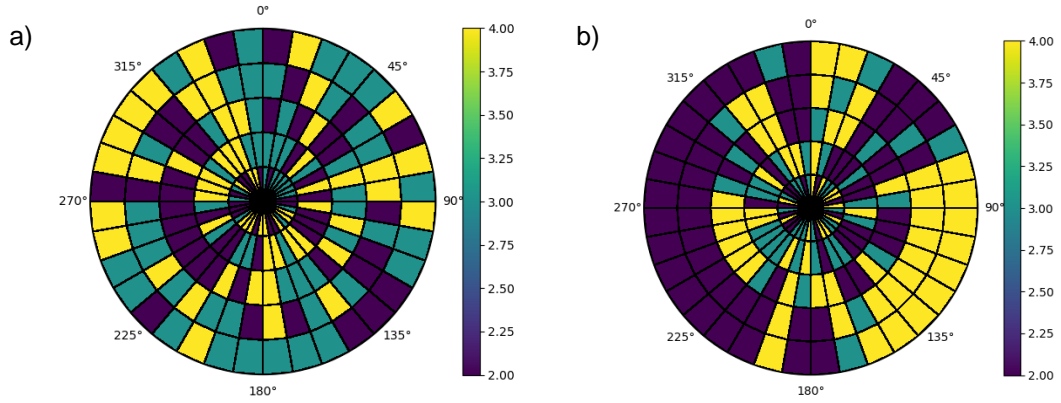


fig. 4.16 visualization of two-dimensional model on a rotating circular disc

a) during 1000<sup>th</sup> iteration

b) during 10'000<sup>th</sup> iteration

Visual representation presented on fig. 4.16a and 4.16b, as on with previous model represented on fig. 4.9a and 4.10, does not show any visible nor regular patterns. Interestingly enough is the value amounts gathered in cells visible on fig. 4.16b. The again, due to the irregular value addition. In fig. 4.16b the values accumulated on the boundary of the disc. This behavior can be due to the introduced simulated rotational movement. The outcome is not a deciding factor for determining the self-organizing criticality of the model.

#### 4.4.2 Impact examination of changing the randomness of value addition in two-dimensional model on a rotating circular disc

As described in points 3.4.2.1 at each iteration a value of 1 is added to the system in consistent  $r_0$  and pseudo-random  $\vartheta_j$  location. To further examine the model, value addition was changed by adding the value to pseudo random  $r_i$  and  $\vartheta_j$  location. The following equation describes the process:

$$Z(r_i, \vartheta_j) = Z(r_i, \vartheta_j) + 1, \quad i \in [0, N], \quad j \in [0, M]$$

where:

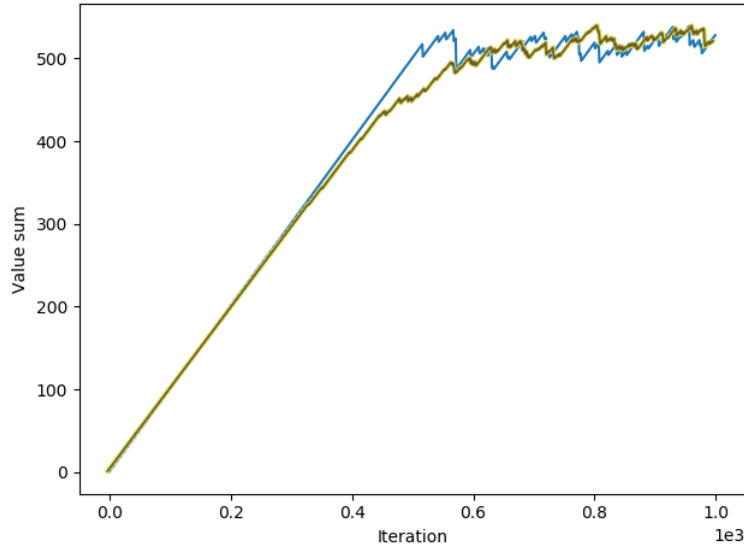
$M$  – maximum user defined integer value

$r_0$  – minimum integer radial value

$\vartheta_0$  – minimum integer angle value

$\vartheta_M$  – maximum integer angle value

$j$  – current y iteration position  
 $i$  – current x iteration position



*fig. 4.17 comparison between two-dimensional on a rotating circular disc when addition is with consistent  $r_0$  (blue) and pseudo-random  $r_i$  (yellow)*

Fig. 4.17 clearly shows that the critical state stability increased when pseudo-random  $r_i$  addition was implemented. The critical point was moved from 500<sup>th</sup> iteration to around 700<sup>th</sup> iteration, then the value sum was steadily increasing, same as on fig. 4.3b. Oscillation peaks are slightly smaller. It can be concluded that the randomness of value addition procures the system to maintain balance as the relocations distribute more equally during simulation. This conclusion even more secures the hypothesis as the studies of self-organization show that the phenomena occurs more frequently when value addition to the system is more randomized. This examination could lead to the development of the inspected model.

## 5. SUMMARY

### 5.1 *Classification of designed model, hypothesis confirmation*

The four main objectives:

- implementation of one-dimensional sandpile model and gathering data,
- implementation of two-dimensional cellular automaton of sandpile model and gathering data,
- designing and implementing two-dimensional cellular automaton with polar coordinates and gathering data,
- designing two-dimensional cellular automaton with polar coordinated and introducing simulated rotational movement and also gathering data.

were achieved. Analyzing data for the one-dimensional and two-dimensional model with a quadratic grid allowed to define a clear control group. Knowing that the control group displays behavioral patterns of self-organizing critical system it was used as a basis on which the designed two-dimensional model on a rotating circular disc was categorized. The system meets the basic self-organizing requirements:

- the system must be open, values or any other variables added must have a possibility to leave the system;
- values or variables addition must be slow and forced;
- the system must be exposed to a local changes that causes a critical instability that tries to be stabilized by some kind of readjustment,

By comparing the obtained data it can be concluded that the designed model can indeed be categorized as a self-organizing critical system on the basis of two points from the self-organizing criticality characterization:

- one small local change in one place impacts the whole system
- after reaching a critical point, the system enters a critical state, trying to sustain stability in that state.

All this conclusions confirm the hypothesis. The implemented two-dimensional model on a rotating circular disc exhibits behavioral patterns of self-organizing critical system and can be characterized as one.

## BIBLIOGRAPHY

- [1] Paul Charbonneau *Natural Complexity: A Modeling Handbook*, Princeton University Press, 2017, Princeton (New Jersey), p. 106 – 128.
- [2] P. Bak, C Tang and K Wiesenfeld, *Self-Organized Criticality: An Explanation of 1/f noise*, Phys. Rev. Lett., Vol. 59 Number 4, 1987.
- [3] Allen B. Downey *Think Complexity*, Green Tea Press, Needham MA, 1<sup>st</sup> Edition, 2008
- [4] Allen B. Downey *Think Complexity 2*, Green Tea Press, Needham MA, 2<sup>nd</sup> Edition, 2011, p. 119 – 126.
- [5] Arka Banerjee *Self-organized criticality in sandpile models*, Illinois Physics Department, Urbana, 2012
- [6] R. Dickman, M. A. Munoz, A. Vespignani and S. Zapperi *Paths to Self-Organized Criticality*, arXiv:cond-mat/9910454v2.
- [7] S. R. Nagel *Instabilities in a sandpile*, Rev. Mod. Phys., Vol. 64 number 1, 1992.
- [8] D. Dhar *Theoretical studies of self-organized criticality*, Physica A 369 (2006) 29-70.
- [9] Matplotlib ver. 3.1.2: <https://matplotlib.org/>, Last accessed on 3.01.2020 (+ versions)
- [10] NumPy ver. 1.18: <https://numpy.org/>, Last accessed on 3.01.2020
- [11] Python ver. 3.7.3: <https://docs.python.org/>, Last accessed on 20.12.2019

## LIST OF FIGURES

3.1 visual representation of two-dimensional model on a circular disc.....	14
3.2 visual representation of value relocation marked as red arrows starting from yellow cell.....	18
4.1 time series of value sum accumulated in the one-dimensional system.....	20
4.2 time series of relocated values in the one-dimensional system.....	20
4.2.1 time series of relocated values in the one-dimensional system for 1'000'000 iterations.....	21
4.3a time series of value sum accumulated in the two-dimensional system for 1000 iterations.....	22
4.3b time series of value sum accumulated in the two-dimensional system for 10'000 iterations.....	22
4.4a time series of relocated values in the two-dimensional system for 1000 iterations.....	23
4.4b time series of relocated values in the two-dimensional system for 10'000 iterations.....	23
4.5 graphical representation of two-dimensional model with quadratic grid after 1000 iterations.....	24
4.6a time series of value sum accumulated in the two-dimensional model on a circular disc for 1000 iterations.....	25
4.6b time series of value sum accumulated in the two-dimensional model on a circular disc for 10'000 iterations.....	25
4.7a time series of relocated values in the two-dimensional system on a rotating circular disc for 1000 iterations.....	26
4.7b time series of relocated values in the two-dimensional system on a rotating circular disc for 10'000 iterations.....	26
4.8 comparison between time series of value sum accumulated in two-dimensional control group model and two-dimensional model on a circular disc for 1000 iterations.....	27
4.9a visual representation of two-dimensional model on a circular disc during 1000 <sup>th</sup> iteration.....	27
4.9b visual representation of control two-dimensional model with quadratic grid during 1000 <sup>th</sup> iteration.....	27
4.10 visual representation of two-dimensional model on a circular disc during 10000 <sup>th</sup> iteration.....	28
4.11a time series of value sum accumulated in the two-dimensional model on a rotating circular disc for 1000 iterations.....	29
4.11b time series of value sum accumulated in the two-dimensional model on a rotating circular disc for 10'000 iterations.....	29

4.12a time series of relocated values in the two-dimensional system on a rotating circular disc for 1000 iterations.....	30
4.12b time series of relocated values in the two-dimensional system on a rotating circular disc for 10'000 iterations.....	30
4.13a density to iteration trend line, amount of relocations during each iteration for 1000 iterations.....	31
4.13b density to iteration trend line, amount of relocations during each iteration for 10'000 iterations.....	31
4.14 comparison between two-dimensional model on a circular disc and two-dimensional model on a rotating circular disc for 1000 iterations.....	32
4.15 comparison between control group and two-dimensional model on a rotating circular disc for 1000 iterations.....	32
4.16a visualization of two-dimensional model on a rotating circular disc during 1000 <sup>th</sup> iteration.....	33
4.16b visualization of two-dimensional model on a rotating circular disc during 10'000 <sup>th</sup> iteration.....	33
4.17 comparison between two-dimensional model on a rotating circular disc when addition is with consistent $r_0$ and pseudo-random $r_t$ .....	34

## 6. APPENDIX A: One-dimensional model python code

```
1. import numpy as np
2. import matplotlib.pyplot as plt
3.
4. N = 50
5. E = 0.1
6. critical_slope = 5.
7. n_iter = 1000000
8.
9. sand = np.zeros(N)
10. tsav = np.zeros(n_iter)
11. mass = np.zeros(n_iter)
12.
13. for iterate in range(0, n_iter):
14.     move = np.zeros(N)
15.
16.     for j in range(0, N - 1):
17.         slope = abs(sand[j+1] - sand[j])
18.         if slope >= critical_slope:
19.             print("UNSTABLE")
20.             avrg = (sand[j] + sand[j+1])/2.
21.             move[j] += (avrg - sand[j])/2.
22.             move[j+1] += (avrg - sand[j+1])/2.
23.             tsav[iterate] += slope/4.
24.
25.     if tsav[iterate] > 0:
26.         sand += move
27.
28.     else:
29.         j = np.random.random_integers(0,N-1)
30.         sand[j] += np.random.uniform(0,E)
31.
32.     sand[N-1] = 0.
33.     mass[iterate] = np.sum(sand)
34.
35.     print(f"{iterate}, mass: {mass[iterate]}, sand: {sand}")
36.
37.
38. plt.plot(range(0,n_iter),mass)
39. plt.ylabel("Value Sum")
40. plt.xlabel("Iteration")
41. plt.ticklabel_format(style='sci', axis='x', scilimits=(0,0))
42. plt.show()
43. plt.plot(range(0,n_iter),tsav)
44. plt.ylabel("Relocated value sum")
45. plt.xlabel("Iteration")
46. plt.ticklabel_format(style='sci', axis='x', scilimits=(0,0))
47. plt.show()
48.
```



## 7. APPENDIX B: Two-dimensional model with a quadratic grid python code

```
1. import numpy as np
2. import matplotlib.pyplot as plt
3. from collections import Counter
4.
5. MAX = 4 # maximum heigght of sandpile
6. width = 50
7. height = 50
8. SANDPILE = np.zeros((width, height)) # array generation
9. NUM_ITER = 10000
10. print(SANDPILE)
11. mass_whole = 0
12. mass_during_iteration = []
13. dislocated_count = 0
14. when_discolaction = []
15.
16. iteration_array = []
17. for i in range(0, NUM_ITER, 1):
18.     iteration_array.append(i)
19.
20. for i in range(NUM_ITER):
21.     print(f"iteration: {i}")
22.     SANDPILE[int(width/2)][int(height/2)] += 1
23.     mass_whole += 1
24.     print(f"{i}: mass {mass_whole}")
25.     for x in range(0, width, 1):
26.         for y in range(0, height, 1):
27.             print(f"cell: ({x}, {y})")
28.
29.             if SANDPILE[x][y] >= MAX:
30.                 SANDPILE[x][y] = SANDPILE[x][y] - 4
31.                 dislocated_count += 1
32.                 when_discolaction.append(i)
33.
34.                 if x + 1 < width:
35.                     SANDPILE[x + 1][y] += 1
36.                 else:
37.
38.                     mass_whole -= 1
39.
40.                 if x - 1 >= 0:
41.                     SANDPILE[x - 1][y] += 1
42.                 else:
43.
44.                     mass_whole -= 1
45.
46.                 if y + 1 < height:
47.                     SANDPILE[x][y + 1] += 1
48.                 else:
49.
50.                     mass_whole -= 1
51.
52.                 if y - 1 >= 0:
53.                     SANDPILE[x][y - 1] += 1
54.                 else:
55.
56.                     mass_whole -= 1
57.
```

```

58.
59.         print(SANDPILE)
60.         mass_during_iteration.append(mass_whole)
61.
62.     print("Final Sandpile")
63.     print(SANDPILE)
64.
65.     plt.plot(iteration_array, mass_during_iteration)
66.     plt.ticklabel_format(style='sci', axis='x', scilimits=(0,0))
67.     plt.xlabel("Iteration")
68.     plt.ylabel("Value sum")
69.     plt.show()
70.
71.     lists = sorted(Counter(when_discolaction).items())
72.     when_discolaction, dislocated_count = zip(*lists)
73.     plt.bar(when_discolaction, dislocated_count)
74.     plt.ticklabel_format(style='sci', axis='x', scilimits=(0,0))
75.     plt.xlabel("Iteration")
76.     plt.ylabel("Relocated value sum")
77.     plt.show()

```

## 8. APPENDIX C: Two-dimensional model with a circular rotating disc python code

```
1. from random import randint
2. import numpy as np
3. import matplotlib.pyplot as plt
4. import seaborn as sns
5. from collections import Counter
6.
7. class Sandpile:
8.     """
9.     Class representing sandpile model for Self-Organized
    Criticality
10.
11.
12.     number_radial = number representing r values (must be odd
    number)
13.     umber_angles = numer representing theta values
14.     array = array representing sandpile with r and theta
15.     max_peak = maximum sand grains on position [r][theta]
16.     topple_count = sums how many topples occurred during
    simulation
17.     mass_count = sums whole mass gathered during simulation
18.     mass_fallen_count = sums mass that was thrown out outside
    disk
19.     mass_left_count = sums mass left on disk
20.     mass_when_iteration = array consisting of mass during every
    iteration
21.     when_topple = array consisting of iteration when topple
    occurred
22.
23.     angles_array = angles array for plotting (required by
    matplotlib)
24.     radial_array = radial array for plotting (required by
    matplotlib)
25.     """
26.
27.     #initialization of sandpile
28.     def __init__(self, number_radial, number_angles, max_peak,
    force=False, addition_random=False):
29.         """
30.         Initialisation function for Sandpile class
31.         :param number_radial:
32.         :param number_angles:
33.         :param max_peak:
34.         """
35.         self.number_radial = number_radial
36.         self.number_angles = number_angles
37.         self.array = np.zeros((self.number_radial,
    self.number_angles))
38.         self.max_peak = max_peak
39.         self.topple_count = 0
40.         self.mass_count = 0
41.         self.mass_fallen_count = 0
42.         self.mass_left_count = 0
43.         self.mass_when_iteration = []
44.         self.when_topple = []
45.
46.         self.number_of_simulations = 0
47.
```

```

48.         self.force = force
49.         self.addition_random = addition_random
50.
51.         self.angles_array = np.linspace(0, 2 * np.pi,
number_angles + 1)
52.         self.radial_array = np.linspace(0, number_radial,
number_radial + 1)
53.
54.     def count_mass_left(self):
55.         """
56.         Function for counting how much mass is left on disk
57.         :return:
58.         """
59.         self.mass_left_count = int(np.sum(self.array))
60.
61.     def add_grain(self, radial_position):
62.         """
63.         Adds grain to chosen radial position and random angle
position
64.         :param radial_position:
65.         :return:
66.         """
67.
68.         self.mass_count += 1
69.         if self.addition_random:
70.             self.array[randint(0, self.number_radial -
1)][randint(0, self.number_angles - 1)] += 1
71.         else:
72.             self.array[radial_position][randint(0,
self.number_angles - 1)] += 1
73.
74.     def topple_simulate_centrifugal_force(self,
radial_position, angle_position, iteration, direction='right'):
75.
76.         #gather data
77.         self.topple_count += 1
78.         self.when_topple.append(iteration)
79.
80.         #execute topple with force simulation
81.         taken_grains = 3
82.         self.array[radial_position][angle_position] -=
taken_grains
83.
84.         #topples right
85.         if angle_position == self.number_angles - 1:
86.             self.array[radial_position][0] += 1
87.         else:
88.             self.array[radial_position][angle_position + 1] +=
1
89.
90.         #topples down
91.         if radial_position < self.number_radial - 1:
92.             self.array[radial_position + 1][angle_position] +=
1
93.         else:
94.             self.mass_fallen_count += 1
95.
96.         #topples skos
97.         if radial_position < self.number_radial - 1:
98.             if angle_position == self.number_angles - 1:
99.                 self.array[radial_position + 1][0] += 1

```

```

100.         else:
101.             self.array[radial_position + 1][angle_position
+ 1] += 1
102.         else:
103.             self.mass_fallen_count += 1
104.
105.     def topple(self, radial_position, angle_position,
iteration):
106.         """
107.         Topple function, gathers data, strips current sandpile
location, distributes taken grains to 3 nearby locations
108.         :param radial_position:
109.         :param angle_position:
110.         :param iteration:
111.         :return:
112.         """
113.         #gather data
114.         self.topple_count += 1
115.         self.when_topple.append(iteration)
116.         #execute topple
117.         taken_grains = 3
118.         self.array[radial_position][angle_position] -=
taken_grains
119.
120.         #one grain topples downwards
121.         if radial_position < self.number_radial - 1:
122.             self.array[radial_position + 1][angle_position] +=
1
123.         else:
124.             self.mass_fallen_count += 1
125.
126.         #one grain topples LEFT
127.
128.         if angle_position == 0:
129.             self.array[radial_position][self.number_angles - 1]
+= 1
130.         else:
131.             self.array[radial_position][angle_position - 1] +=
1
132.
133.         #one grain topples RIGHT
134.
135.         if angle_position == self.number_angles - 1:
136.             self.array[radial_position][0] += 1
137.         else:
138.             self.array[radial_position][angle_position + 1] +=
1
139.
140.     def check_pile(self, iteration):
141.         """
142.         Function checking every sandpile location, if grains of
sand exceed max topple starts
143.         :param iteration:
144.         :return:
145.         """
146.
147.         for r in range(0, self.number_radial, 1):
148.             for theta in range(0, self.number_angles, 1):
149.
150.                 if self.array[r][theta] < self.max_peak:
151.                     self.array[r][theta] = self.array[r][theta]

```

```

152.
153.         else:
154.             if self.force:
155.
156.                 self.topple_sumulate_centrifugal_force(r, theta, iteration)
157.             else:
158.                 self.topple(r, theta, iteration)
159.
160.     def simulate(self, number_of_simulations):
161.         """
162.         Main function, starts sandpile simulation
163.         :param number_of_simulations:
164.         :return:
165.         """
166.         self.number_of_simulations = number_of_simulations
167.         for iteration_num in range(0, number_of_simulations,
168.                                     1):
169.             self.add_grain(0)
170.             self.check_pile(iteration_num)
171.             self.mass_when_iteration.append(self.mass_count -
172.                                             self.mass_fallen_count)
173.             print(self.array)
174.
175.     def plot(self, type='sandpile', bandwidth=1):
176.         """
177.         Plotting function
178.         :param type:
179.         :return:
180.         """
181.         #plot sandpile after simulation
182.         if type == 'sandpile':
183.             fig, ax =
184.                 plt.subplots(subplot_kw=dict(projection='polar'))
185.             cb = ax.pcolormesh(self.angles_array,
186.                               self.radial_array, self.array, edgecolors='k', linewidths=1)
187.             ax.set_yticks([])
188.             ax.set_theta_zero_location('N')
189.             ax.set_theta_direction(-1)
190.             plt.colorbar(cb, orientation='vertical')
191.             # plt.ticklabel_format(style='sci', axis='x',
192.             scilimits=(0, 0))
193.
194.             plt.savefig(f'plot_{self.number_of_simulations}_sandpile_{self.f
195.                         orce}.png')
196.
197.             plt.show()
198.
199.         #plot iteration / mass of left pile on plate
200.         if type == 'mass':
201.             simulation_array = []
202.             for i in range(0, self.number_of_simulations, 1):
203.                 simulation_array.append(i)
204.
205.             plt.plot(self.mass_when_iteration)
206.             # plt.title("Left Mass on pile during iterations")
207.             plt.xlabel("Iteration")
208.             plt.ylabel("Value sum")
209.             plt.ticklabel_format(style='sci', axis='x',
210.                                 scilimits=(0, 0))

```

```

203.     plt.savefig(f'plot_{self.number_of_simulations}_mass_{self.force
    }.png')
204.         plt.show()
205.
206.         #plot iteration / topple
207.         if type == 'topple':
208.             lists = sorted(Counter(self.when_topple).items())
209.             when_topple, topple_count = zip(*lists)
210.
211.             print("TOPPLE COUNT")
212.             print(topple_count)
213.             print("TOPPLE COUNT LEN")
214.             print(len(topple_count))
215.
216.             print("LISTS")
217.             print(lists)
218.
219.             plt.bar(when_topple, topple_count)
220.             plt.xlabel('Iteration Number')
221.             plt.ylabel('Relocated value sum')
222.             plt.ticklabel_format(style='sci', axis='x',
    scilimits=(0, 0))
223.     plt.savefig(f'plot_{self.number_of_simulations}_topple_{self.for
    ce}.png')
224.         plt.show()
225.
226.         if type == 'histogram':
227.
228.             lists = sorted(Counter(self.when_topple).items())
229.             when_topple, topple_count = zip(*lists)
230.
231.
232.             if bandwidth == 1:
233.                 plt.hist(self.when_topple, color='blue',
234.                     bins=int(180 / bandwidth))
235.             else:
236.                 plt.hist(self.when_topple, color='blue',
    edgecolor='black',
237.                     bins=int(180 / bandwidth))
238.             plt.ylabel("Relocation amount")
239.             plt.xlabel("Iteration")
240.             plt.show()
241.
242.             if type == 'pdf':
243.
244.                 if bandwidth == 1:
245.                     sns.distplot(self.when_topple, hist=True,
246.                         kde=True, bins=int(180 / bandwidth),
247.                             color='blue',
248.                             kde_kws={'linewidth': 3})
249.                 else:
250.                     sns.distplot(self.when_topple, hist=True,
251.                         kde=True, bins=int(180 / bandwidth),
252.                             color='blue',
253.                             hist_kws={'edgecolor': 'black'}, kde_kws={'linewidth': 3})

```

```

254.
255.     def analyze_data(self, bandwidth=1):
256.         """
257.         Function for analyzing data and executes plotting.
258.         :return:
259.         """
260.         data = {"Topple Count": self.topple_count, "Fallen
mass": self.mass_fallen_count}
261.         print(data)
262.
263.         self.plot()
264.         self.plot(type='mass')
265.         self.plot(type='topple')
266.         self.plot(type='histogram',bandwidth=bandwidth)
267.         self.plot(type='pdf',bandwidth=bandwidth)

```