NAME: ETHAN SHANAHAN

I.D: 19269064

SUPERVISOR: DR. DAVID CORCORAN

COURSE: B.Sc. MATHEMATICS AND PHYSICS

YEAR: FOURTH YEAR

DEPARTMENT: PHYSICS

PROJECT TITLE: CONTROLLING SELF-ORGANISED CRITICALITY

DATE: 14th APRIL 2023

Table of Contents

Acknowledgments

It is my pleasure to express a message of immense thanks to my thesis supervisor, Dr. David Corcoran. Without his utmost dedication to his work, I would not have had this opportunity to work on a project I so cherished. This has been instrumental in the shaping of my personal academic journey. His instruction and encouragement were pivotal in ensuring the prolific success of the project and in guiding my assimilation into the University of Limerick. As I reflect on my time at the University of Limerick, and particularly on this last, strenuous year, I would like to send a heartfelt acknowledgement to my fellow students, who are too many to name, insofar as I owe to them my emotional stability and fragility. Their criticisms, be they constructive or slanted, are truly appreciated. Finally, I would like to share my deepest gratitude with my family. Their undying faith in me is a shocking wonder that inspires me daily. This project would not have come to fruition without my foundation, which is their love and support.

I am exceedingly grateful to all those who have played a part in this journey. Through their blessings, I have been able to strive to achieve my goals, and I look forward to continuing to pursue many an endeavour with them.

Abstract

In this project, a new Python library for cellular automaton simulations is presented. The new library provides various built-in functionalities that allow the user to analyse and visualise their simulations. A focus is put on two well-studied models, the BTW and OFC models. The new library has complete implementations for both of these but also offers excellent extensibility so that many more models can be simulated without much effort. The flexibility of the library is demonstrated.

With this new library, two control strategies are investigated on the OFC model. The control strategies consist of a feedback loop and various logical conditions that produce an external perturbation between events. The aim of the control strategies is to simply have a non-trivial effect, since the ability to influence the behaviour of these systems opens up many new avenues of research. These experiments show a decrease in the frequency of large events, which is a very promising result.

Chapter 1: Introduction

1.1 Background

Chaos, by definition, is something intangible and worrisome. Disorder is perfectly conducive to disaster, and unfortunately, it is the natural state of the universe. Time has proven a relentless harbinger of phenomena both beautiful and destructive, and so it should be of no surprise that humanity has made it a calling to seek answers within this chaos. There is a slight nuance here that is worth mentioning. In "chaos theory", the phenomena or systems studied are usually deterministic. How exactly this compares to reality is unknown, but what chaos theory tells us is that even deterministic systems can produce results that appear remarkably random on the surface. What then does it matter if the system, or in fact the universe, is deterministic or stochastic? If, regardless of this attribute, one can employ methods of analysis to qualitatively predict and create, then perhaps the line between what is certain and what is not is likewise intangible.

Take, for example, the simple pendulum, which is perfectly described by Newton's laws of motion. One can exactly predict the pendulum's position at any point in time. Now consider the double pendulum. How different is this system really? Once there was a single bob; now there are two bobs. The system is conceptually basic but impossibly complex in practise. I remind you of this problem to demonstrate the nature of complex physical systems. The crucial component is interconnected parts, but clearly, this is not meant to imply that some laborious weaving must be performed to exemplify chaos. Oftentimes, the exact opposite is the case. Systems, from the most convoluted to the most elementary, can display random behaviour as long as their components can communicate. This manifests as nonlinearity, feedback loops, networks, self-similarity, fractals, probability distributions, and so on.

Natural phenomena persistently pose dilemmas for the concerned scientist, and yet complete solutions to most remain elusive forever. Perhaps there is no greater problem

than the inner workings of nature. In some sense, everything can be reduced to a raw series of events produced by the intrinsic laws of our universe. Hence the scope of this problem is inconceivably broad, and one does not need to think too hard before finding problems with the world. Consider a town set at the foot of a tall, snowy mountain. Every day, a sprinkling of snow dusts the land, and atop the mountain, a steep mound accumulates. Eventually, the snow piled high will slip, but who can say what will happen then? Maybe the inhabitants of our little town are lucky, and the snow comes to rest against a crevice. Then again, maybe their luck has just run out and that pile of snow triggers an avalanche that ploughs through their beloved homes. I hope the motivation to study such phenomena is clear.

Study leads to understanding, and understanding leads to control. I hope to investigate phenomena like these, particularly self-organised critical (SOC)[1] systems, to determine how effectively a control strategy can influence their behaviour. So, how does one begin to understand such complex systems that result in random behaviour? Chaos theory is so inspiring because it is really about the discovery of hidden patterns lurking behind the guise of stochasticity. Today, a wealth of techniques exist to analyse these systems, such as Poincaré maps, bifurcation theory, phase space embedding, Lyapunov exponents, perturbation theory, and many more [10]. For this project, however, I will take a more numerical approach to the analysis. This is for two reasons. First, analytical solutions and approximations are not only challenging but also fresh from the literature when it comes to self-organised criticality and cellular automata, which I will explain shortly. Second, the numerical results will allow for direct comparisons to the original papers that demonstrated models of interest and for succinct representations of control ramifications.

[1]Note that this acronym often stands alone to mean "self-organised criticality", being the property that such systems hold.

1.2 Self-Organised Criticality

This project is concerned with SOC systems, and it is a well-placed concern. I have already introduced one phenomenon that can be modelled this way, that being the occurrence of avalanches. Indeed, the word avalanche is often used in the literature surrounding this topic to refer to events that are completely unrelated to snowy mountainsides. These types of systems are a subset of dynamical systems that must have an attractor at a critical point. This says a few things about the properties of the system, and it should become apparent that many natural phenomena may be modelled accordingly.

Our illustrative example is dynamic in the sense that it takes the interactions between many snowflakes to form one homogenous avalanche. That statement reveals another property of complex systems in general. "Emergence" refers to grand behaviours emerging from the smaller constituent parts of the system. That is to say, the action of the whole is greater than the sum of the actions of the parts [12]. The critical point of the snow-capped mountain system is the exact moment when not enough snow becomes too much snow, and an event[2] occurs. The critical point, in this sense, is similar to the traditional critical point found in thermal physics when matter undergoes a phase transition, but there is something fundamentally different here. The point must also be an attractor, a mathematical construct that is often studied in chaos theory. A system that contains an attractor is one that will evolve over time into a certain state or set of states. Although it may seem counterintuitive for a critical point, a dramatic change in the system's state, to be an attractor, a state the system tends towards, this is nonetheless the type of system being considered. The argument for this stems from the success of these models in simulating many natural phenomena. It can be seen intuitively that there are in fact many systems that may exhibit SOC. In complex physical systems of the natural world, there is often a slow driving force, like the gradual snowfall at high

[2]Event will be used to refer to what happens in a system once a threshold is met. An avalanche, earthquake, or market crash are all examples of events that occur in different systems.

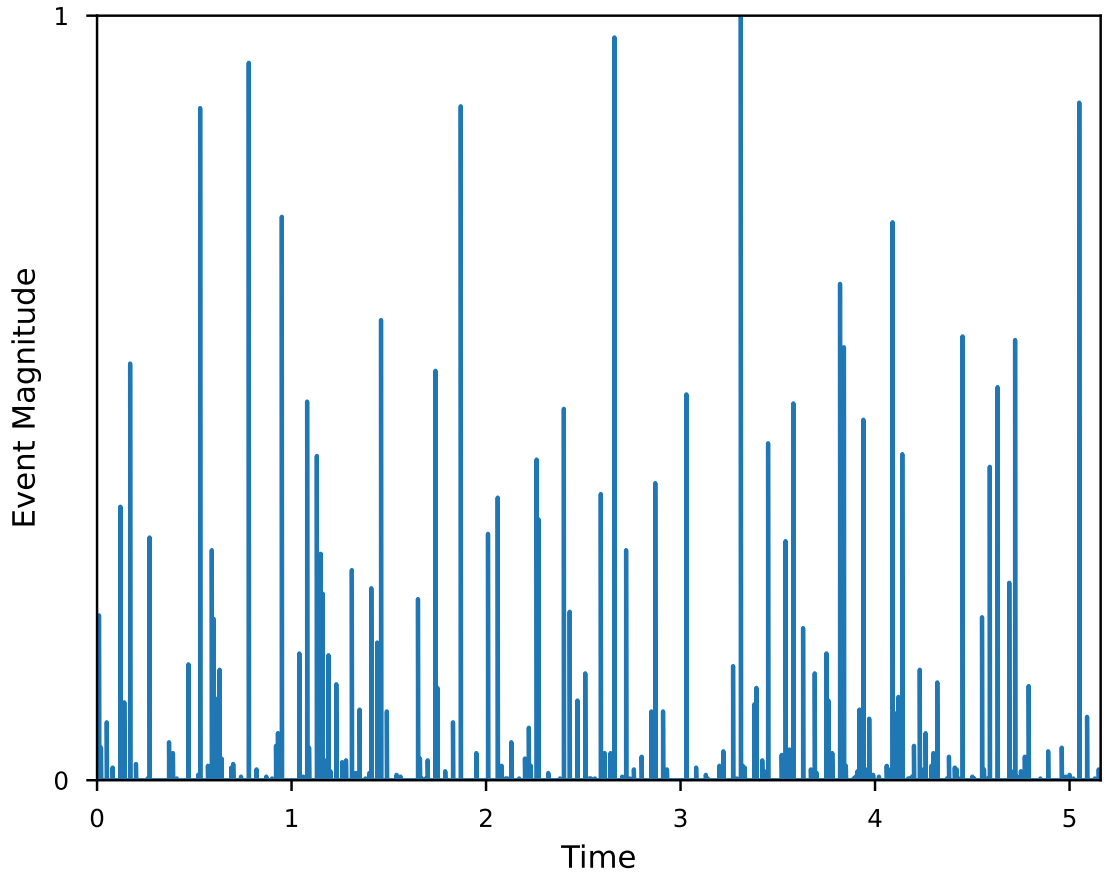

Figure 1.1: The time-series of measured event magnitudes from a self-organised critical system.

altitudes, which perturbs the system. While at the critical point, this small perturbation sends the system into instability, where it evolves back to stability right on the edge of chaos. Figure 1.1 shows the time-series of a self-organised critical system. Clearly, the events don't seem to follow a simple pattern.

1.3 Cellular Automata

Cellular automata are tools that facilitate the numerical analysis of highly dynamic nonlinear systems, particularly ones in which there are a vast number of discrete parts that each possess a well-defined state. See Figure 1.2 for a typical example of the cellular automata used in this project. Due to the nature of these systems, it can be challenging for an algorithm to account for the interplay of dependencies on many changing parts. A cellular automaton atomises the system into an array of cells. Each cell takes on a value,

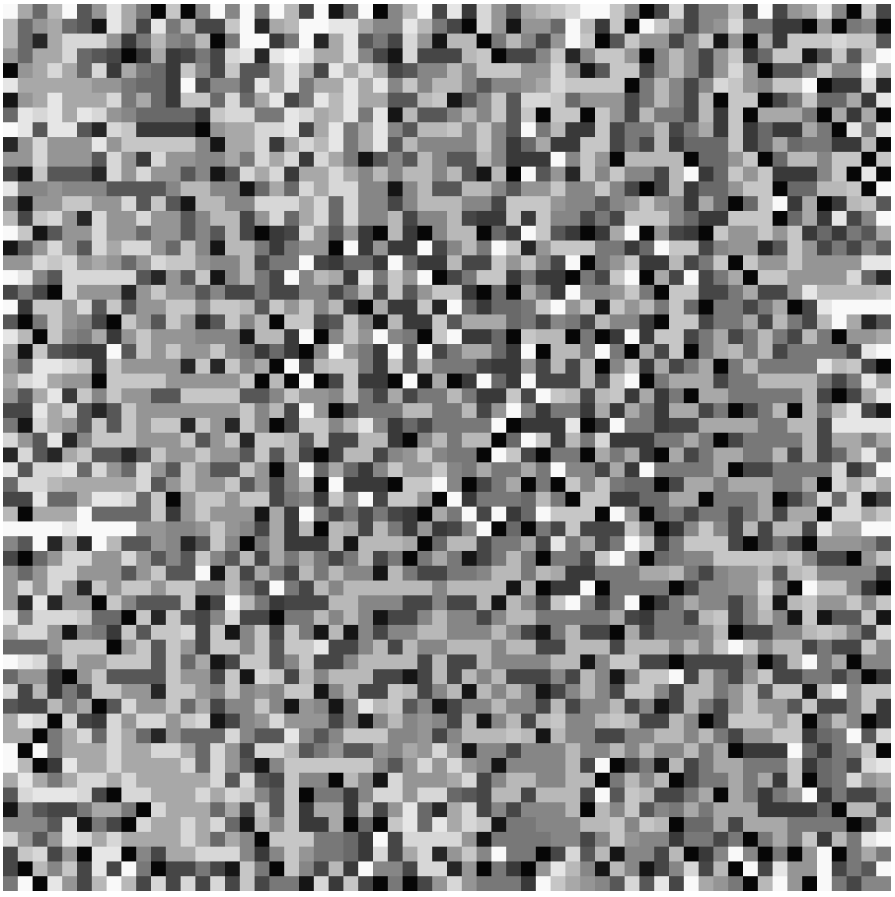

Figure 1.2: A 60×60 array of cells with random values between 0 and 3. The brighter the shade of the cell, the higher it's value.

and together they comprise the array. The system is entirely represented by the array, coupled with a set of rules that govern the initial state of each cell, the behaviour of edge cases, and how the dynamics work between time-steps. There are countless ways to customise a cellular automaton to more closely mirror the real system. The dimensions of the array can be anything the computer's memory can handle. Most commonly used to simulate physical systems are 2-dimensional and 3-dimensional arrays. One-dimensional arrays have also received a lot of attention, and even higher-dimensional arrays [7] can be used to simulate complex models that may or may not be dependent on spatial or temporal variables.

The simulation evolves across time-steps based on update rules. Each cell is subjected to this rule to determine what effect it will have on the rest of the system. Critically, this must be synchronised for every cell in the system. The control portion of

the project is also facilitated by cellular automata. By carefully constructing the code such that each state of the system can be detected continuously, a feedback response can be applied between time-steps. The simulation would be able to react to itself and introduce excess perturbations with the intention of influencing the subsequent states.

Cellular automata are often used to model complex systems, such as biological or physical processes, and have been applied to a wide range of fields, including physics, chemistry, biology, computer science, and economics. They continue to be an active area of research, and their versatility and simplicity make them a valuable tool for exploring complex systems. Due to the nature of self-organised critical systems, cellular automata are ideal simulations. However, despite their theoretical aptness, practically, these solutions can be highly computationally intensive. Care must be taken to balance the ideal model with realistic programmatic implementations.

1.4 Aims and Objectives

This project is strongly motivated and ambitious. The first objective of the project is the development of a flexible and expandable script that models dynamical systems using cellular automata. Existing applications and libraries were found to be limited in functionality and lacking current support, so the decision was made to create something that could potentially be deployed as an openly accessible tool. The second objective is to verify the newly written script on pertinent models in SOC research. The BTW model from Bak *et al.* [3] and the OFC model from Olami *et al.* [18] are used as a baseline to compare the basic performance of the simulation and to highlight the measurable features of the model. The third objective is to apply the control mechanism of the script. The subsequent simulations will be contrasted with the established behaviour of the OFC model. A range of control techniques will be considered with the aim of reducing the size of large events. The results will be analysed to determine an effective strategy in the effort to control the system's long-term traits.

Chapter 2: Literature Review

2.1 The Seminal Paper and the BTW Model

In 1987, Per Bak, Chao Tang, and Kurt Wiesenfeld published their paper titled "Self-organized criticality: An explanation of the $1/f$ noise" [3] in Physical Review Letters. In this work, the researchers exhibited the first dynamical system to display self-organised criticality. This was a ground-breaking achievement and endowed the scientific community with a new field of research. SOC is put forward as an explanation for the occurrence of $1/f$ noise as detected in the studies of countless natural phenomena. Concerns have been raised over this claim [13], but the continuing research keeps providing new insights [23] [17] into the still emerging field of complex systems.

Although the exact characteristic properties of SOC are yet unknown, the following features are typical: self-organised criticality is the property of highly non-linear dynamical systems where an attractor is located at the critical point. This has the interesting effect of differentiating these critical points from those ordinarily seen in other areas of science. Without this property, the critical point, if any, can only be reached by manually tuning the system towards it, such as by increasing the temperature of matter to cause a phase transition. In SOC, there exists a natural law that guides the system towards criticality, so there is no need for external tuning. Between events, the system is considered minimally stable. This implies that although the system is stable, it becomes unstable even for small perturbations. Upon becoming unstable, an event takes place where the system is relaxed according to the update rules set out by Bak. These rules evolve the system to yet another minimally stable state, so the system remains critical. For anything to happen, there must be some mechanism by which the minimally stable system is perturbed. This is usually a naturally occurring, slow driving-force that has no bearing on the relatively rapid relaxation of the instability it causes. Another mechanism is the purposeful control of the system by external factors. This is not to be confused with tuning, however, since the principal behaviour of the SOC system is not dependent

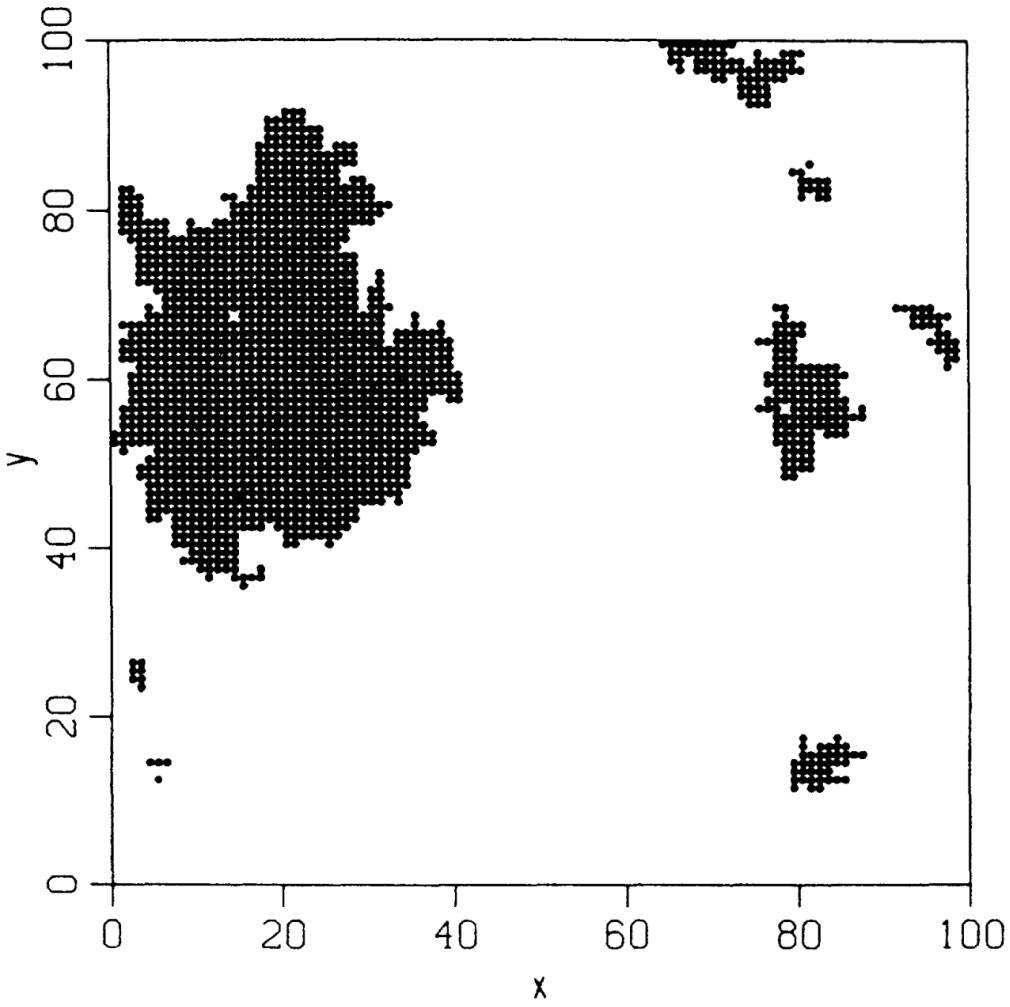

Figure 2.1: Reproduced from Bak *et al.* [4]. The clusters of black dots outline typical event sizes resulting from several perturbations for a 100×100 array. Each cluster arose from different events.

on this kind of perturbation. This type of influence was not studied by Bak *et al.* and has yet to receive a worthy exploration in the literature.

From a quantitative standpoint, the authors used cellular automata to simulate their model, and they measured the sizes of events by simply counting the number of cells involved. See Figure 2.1 for a visual demonstration of events in a cellular automaton array. The distribution of these sizes was found to match a power law relationship of the form $P(s) \sim s^{-\alpha}$. The simplicity of the system comes from the minimal instructions for how elements of the system react. The cellular automaton used in their paper was analogous to a pile of sand, where the value of each cell was an integer representing

the slope, or potential energy, of the sand at that point. Bak defined the update rules to trigger when the value of a cell z exceeds a threshold value K, as follows:

$$z_t(x, y) \rightarrow z_{t-1}(x, y) - 4 \tag{2.1}$$

$$z_t(x \pm 1, y \pm 1) \rightarrow z_{t-1}(x \pm 1, y \pm 1) + 1 \tag{2.2}$$

where $z(x, y)$ is the value of the cell at position (x, y) in a square grid of edge length L. The discrete time-steps of the algorithm are represented by t. These equations then describe how to update a cell and its 4 nearest neighbours $(x \pm 1, y \pm 1)$. This update is performed synchronously on every cell in the grid for which $z > K$. As it would later be proved by Dhar [8] in 1999, the updates need not be performed synchronously in this model. This led him to construct the more formal Abelian Sandpile Model, named after the abelian group of commutative operations in mathematics. Various qualities have since been investigated [21], such as an identity element, weak convergence, and an extended sandpile model for the real domain.

The system governed by those few laws above was investigated by Bak *et al.* with the aim of discovering the generic power-law behaviour of $f^{-\alpha}$. Its appearance would lend credence to self-organised criticality as a fundamental mechanism behind naturally occurring fractals. The power-law relationship is a scale-invariant relationship, and one essential attribute of fractals is their self-similarity on arbitrary scales. The connection between the two is then conveniently explained if natural dynamical systems can be shown to tend towards critical states without the need for external tuning. According to the researchers, this generalisation can predict the temporal and spatial power-law correlations of systems that have yet lacked any satisfying explanations: systems like turbulence, landscape formations, and countless others. Current research has extrapolated the model with great success while maintaining the intrinsic purity of the model.

2.2 Seismicity and the OFC Model

Earthquake modelling is an important area of research that helps scientists better understand the dynamics of seismic activity. If successful, vulnerable civilisations can be identified and measures taken to mitigate their risks. Since the earthquakes themselves can never be precisely predicted, every bit of evidence is crucial for affirming probabilistic models.

The Burridge-Knopoff model is a system of differential equations that predicts the behaviour of earthquakes with a high degree of accuracy and can be experimentally demonstrated. It acts as a simplified model of the behaviour of fault systems, where earthquakes are caused by the accumulation and release of stress along a fault line [6]. The experimental model consists of a series of spring-block systems that are arranged in a two-dimensional lattice lying on a frictional surface. See Figure 2.2. Each spring-block system represents a section of a fault, and the springs carry the elastic forces that build up stress. The system is driven by a constant force that represents how the tectonic plates shift, applying more and more pressure at the interface. The stress on each spring is measured based on its deformation with respect to the neighbouring blocks. If the stress exceeds a certain threshold, the block slips across the frictional surface before quickly sticking again, in what is aptly referred to as stick-slip motion. As this happens to one block, the energy stored in the springs is released, causing a chain reaction through nearby spring-block systems. The energy released by each earthquake is distributed to the neighbouring blocks, causing them to deform and build up stress for the next earthquake.

The Olami-Feder-Christensen (OFC) model [18] is a cellular automaton model based on the Burridge-Knopoff model. It too simulates the behaviour of earthquakes on a two-dimensional lattice. Each cell in the grid has a stress value that is updated at each time step based on the stress values of its neighbouring cells. If the stress at a cell exceeds a certain threshold, it is considered critical and triggers a relaxation of that cell, and the

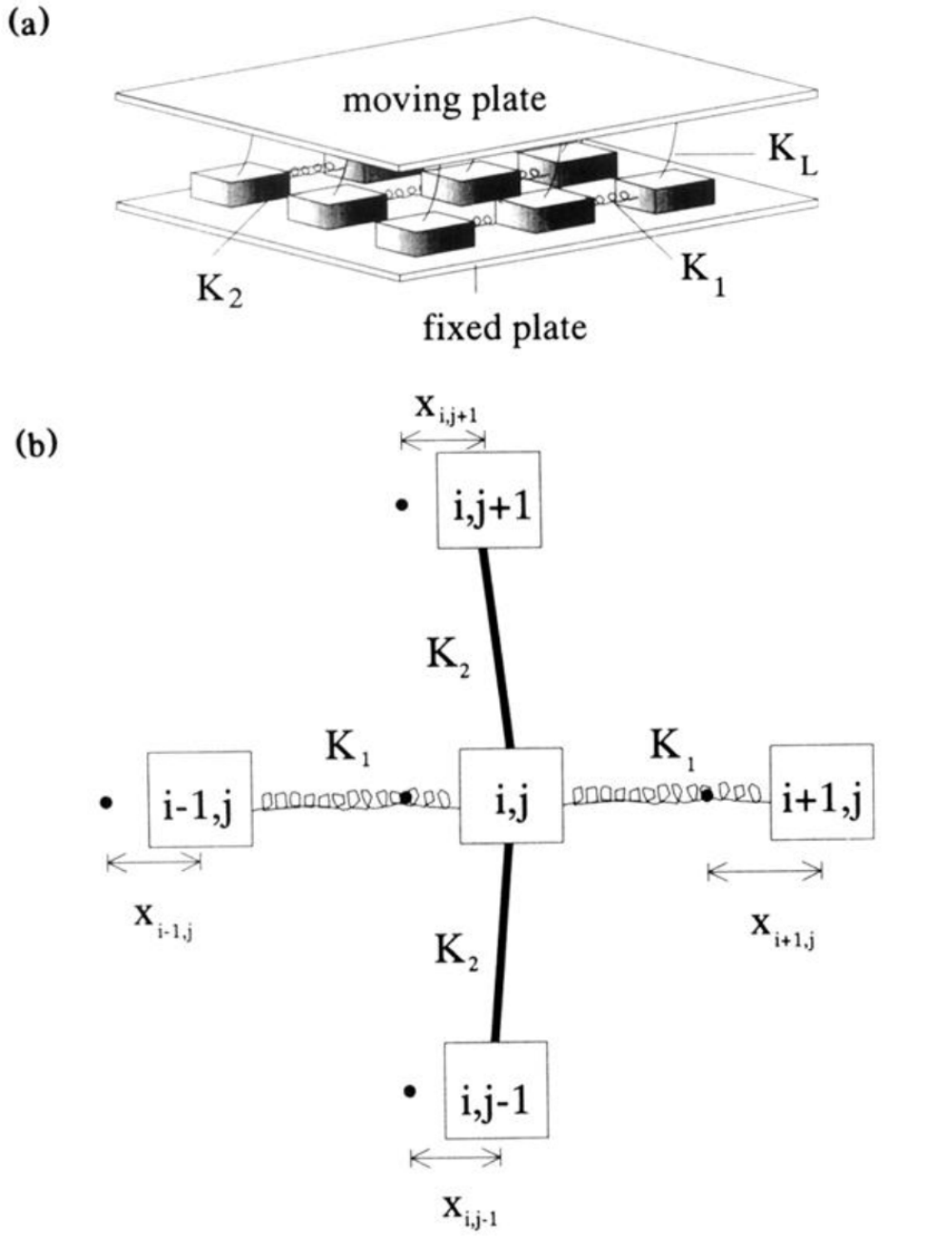

Figure 2.2: Reproduced from Olami *et al.* [18]. The geometry of the spring-clock model. *K*'s are various elastic constants.

energy is redistributed to its four nearest neighbours. The redistribution of energy can trigger more relaxations, leading to a cascade of events much like an earthquake. This all sounds very similar to the BTW model. In many ways it is; however, there are a few instrumental distinctions. First, the stress of a critical cell is set to zero when relaxed, rather than being reduced by a fixed amount. Second, the redistribution of stress is proportional to the stress of the critical cell rather than incrementing each neighbour by a fixed amount. Third, the slow-driving perturbation is global rather than local. Fourth, the relaxation is nonconservative. This last difference easily sets it apart from the BTW model. Olami *et al.* showed that the level of conservation has considerable effects on the dynamics of the system. While the BTW model was perfectly conservative in that when 4 was subtracted from a cell, a total of 4 would be added to other cells. The OFC model uses cell values in the real domain to analyse various degrees of conservation where a fraction of the relaxing cell's stress would be lost. This approach tries to be a much more realistic implementation of the theory. Their findings concluded that power law relationships are evident for conservation parameters in the range of real-world measurements. This success has prompted the OFC model to be used in current studies of earthquake behaviour as well as the dynamics of other self-organised critical systems.

2.3 Fractional Noise in Nature

Returning to the wintry landscape, if one measured the sizes of avalanches, both big and small, over a long enough period of time, one would see the infamous $1/f$ noise [22] appear as a probability distribution, or relationship between the size and frequency with which avalanches of that size occurred. More generally, these naturally occurring events may reveal $f^{-\alpha}$ noise, or fractional noise, discussed at length by Mandelbrot and Van Ness [14], who are now famous for their work on fractals. The subject has remained a fascination of both mathematics and art. Fractals are self-similar structures that repeat themselves on different scales, producing intricate patterns and shapes that often closely resemble those found in nature, like snowflakes or coastlines. In the 1970s, Benoit Mandelbrot introduced the concept of fractals and pioneered the field of fractal

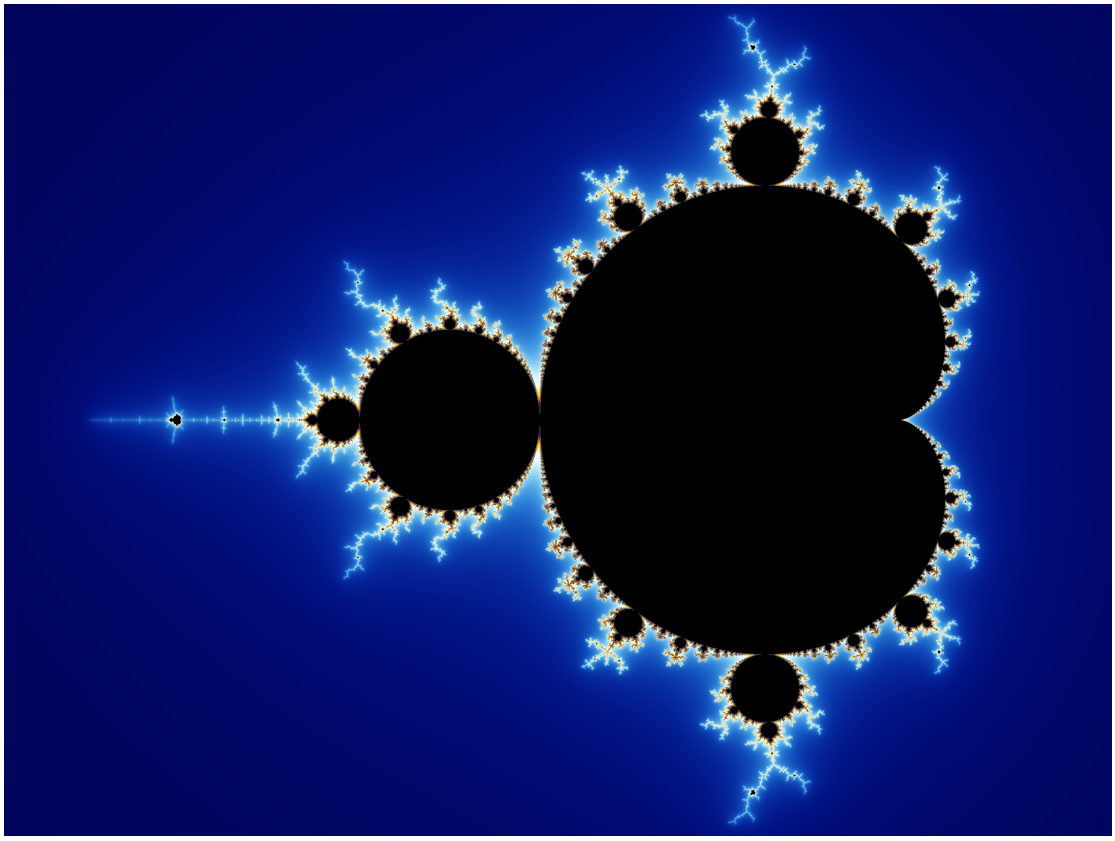

Figure 2.3: A depiction of the Mandelbrot set that was once studied by the eponymous mathematician.

geometry [15]. His work on fractals revolutionised the way we view and understand irregular shapes and patterns in nature. Mandelbrot's name is now immortalised as the Mandelbrot set [16], seen in Fig. 2.3, which is a complex mathematical structure that exhibits self-similarity and infinite complexity.

Boundless evidence of fractional noise, and in particular $1/f$ noise, is demonstrated by Press [20]. The examples provided range from astronomical, as in the light intensity of quasars, to geophysical, as in the ocean current velocities. We sorely lacked an explanation for $1/f$ noise, and it seemed imperative to find one in order to uncover the underlying mechanisms of complex dynamical systems such as the examples herein and, of course, many more. These systems with slow driving forces and local interactions are conducive to chaotic behaviour. They are highly non-linear and seemingly impossible to predict, and yet when one looks at the occurrence of avalanches on mountain sides,

there is a constant that presides over infinite scales. Despite the transience of the inclined snow, avalanches never cease to develop given adequate time. This fact seems obvious from our intuition alone, but mathematics tells us that there is something very special about a chaotic system that somehow organises itself towards these transitionary states between stability and instability.

Possibly most notable among the detections of $1/f$ noise is that found in electronics [25]. There, $1/f$ noise is ubiquitous and is known as "flicker noise", although this term has circulated just as much as "$1/f$ noise" or even "pink noise" has. Almost every electronic device presents flicker noise due to naturally occurring fluctuations in the resistances of materials with micro temperature changes in the environment. It is commonly seen as a low-frequency phenomenon, and this translates into other fields too. The brain is a highly complex system. Networks of neurons influence each other dynamically, responding to their neighbours, generating new results, and cooperating to produce large-scale effects. Recent research has been mounting evidence for self-organising behaviour in brains that leads to characteristic scale-invariance [24]. It is possible for these correlations to be the mechanism behind widespread neuronal activity [19]. The possibility of control in this field could advance our understanding of a relatively unknown part of nature and could lead to improvements in the treatments for conditions of the brain such as epilepsy.

$$P(x) \sim x^{-\alpha} \tag{2.3}$$

Simply put, in terms of the alpine town, The power-law relationship 2.3 predicts that the smallest avalanches occur most frequently and the largest avalanches occur least frequently. This example is a widely extensible analogy but is a specifically fantastic motivator since nobody wants the little town to be destroyed.

2.4 Control

Control theory is a branch of mathematics and engineering that deals with the design and analysis of systems that can be controlled to achieve a desired output. In dynamical systems, control theory is used to design control loops that can stabilise the system, track a reference trajectory, and otherwise regulate the behaviour of the system. Closed-loop control, or feedback, is a mechanism in a control system that takes measurements of the output and uses them to adjust the input. This mechanism must somehow detect the system and compare the measured output to a reference value. The difference between the two is used to compute a control signal that adjusts the input, usually trying to reduce the error between the output and the reference value, thus stabilising the system. In dynamical systems, the models are typically represented by differential equations that describe the evolution of the system over time. The design of feedback loops is then based on mathematical models [11].

Cellular automaton models are a class of dynamical systems that are discrete in time and space. Control in cellular automaton models can be achieved using feedback loops that adjust the rules based on the state of the system or introduce an external influence between unperturbed time-steps. For example, in a model of traffic flow, the state of each cell represents the occupancy of the cell by a vehicle, and the rules determine how the vehicles move from one cell to another. A feedback loop can be used to adjust the rules based on the traffic density to improve the flow of traffic and prevent congestion.

Control theory has many applications in engineering, economics, and other fields where dynamical systems are used to model complex phenomena [2]. It is used in the design of aircraft, automobiles, and other machines and in the control of chemical processes, power systems, and other industrial systems. Feedback loops are an essential component of many control systems, and their design is a key challenge in control theory. The gains of the feedback loop must be carefully tuned to ensure that the system is stable and the response is fast and accurate. Emerging at a blistering rate in this field

is the application of machine learning to adapt the control strategy dynamically. In a model where the components of the complex system are themselves dynamical agents, unique insights have been uncovered with regard to the ability of control to optimise. Social insects have demonstrated impressive collective intelligence by way of adapting and responding to the interactive network of individuals [5]. Traditional control may be superseded by these new artificial intelligence models, which are uniquely autonomous, flexible, and robust.

Chapter 3: Experimental Procedure

3.1 A Novel Software

In this project, experiments are conducted virtually. Self-organised critical systems are studied using cellular automata, which are implemented in computer code. It was important to find a balance between two qualities of the code: efficiency and flexibility. The advantages of simulating these systems with a computer are the ability to gather huge sample sizes quickly and the ability to easily adjust the experimental setup as required. A new programme had to be written to accomplish all of this, for reasons that will become clear. Both Python and MATLAB were considered at first. These are interpreted[3] languages meaning the development process is fast and fluid. Especially for writing a completely new script from the ground up with only limited knowledge of programming beforehand, it was an indispensable feature that expedited the process. Ultimately, Python was chosen over MATLAB for two reasons. Neither language had existing support for all that was required by my project, but Python has better extensibility with its vast supply of community-driven libraries. Although arrays are not even natively supported in Python, the NumPy library[4] seamlessly extends the language, and the Matplotlib library provides data visualisation functionality. The second reason for choosing Python was that it is distributed under an open-source licence. MATLAB is proprietary software, and although it is very popular among physicists, it cannot match the accessibility of Python. Since one aim of this project is to develop software for widespread use, Python is the better choice.

Some attempts were made to source an existing, community-authored library for cellular automata, and a couple of options were found. From PyPi, “cellular-automaton”

[3]Python does some compilation to bytecode behind the scenes but behaves for the most part like a regular interpreted language.

[4]A Python library is a collection of interconnected bundles of code such as functions, classes, modules, etc. They may be used by other Python programmers to easily achieve something that would be rather convoluted in the base language alone.

[9] is a package first released in 2019 and last updated in 2021. It is well optimised and supports n-dimensional arrays. However, it is limiting in the creation of custom update rules and does not support the BTW or OFC models out of the box. "CellPyLib" [1] was released in 2021 and is being actively developed. It only supports 1-dimensional and 2-dimensional arrays, although this would be sufficient. This library, however, is catered towards elementary cellular automata[5] and adapting them for the purposes of this project would be cumbersome. Aside from these challenges, neither library facilitates the control of the system via external strategies or feedback. This missing niche, among the other limitations, is used to motivate the development of a new Python library.

3.1.1 Scope

It is imperative that the scope of this new library be established. In software development, scope refers to the boundaries of a project and the features, functions, and requirements that are included within those boundaries. It is a critical aspect of software development as it helps developers define the project's goals and determine the resources and timeline required to accomplish those goals. Furthermore, a well-defined scope enables developers to prioritise the project's features and functionalities according to their importance, urgency, and complexity. It becomes easier to identify any potential roadblocks and dependencies early in the development process. The importance of scope in developing a programme cannot be overstated, as it directly impacts the project's timeline and overall success.

This library aims to patch what are seen as shortcomings in other libraries and build upon what has been done well. N-dimensional support seems natural. Bak *et al.* primarily performed cellular automata on 2-dimensional and 3-dimensional grids, for instance, and 1-dimensional grids are the most studied in adjacent fields. Initial conditions are treated very differently by researchers who use cellular automata in

[5]Elementary cellular automata are 1-dimensional, binary models ruled by nearest neighbour interactions and are usually solved with look-up tables. They are considerably different from the models used in this project, but fundamentally the same principles apply to both.

different fields, but in this project, there should be little variety. There is certainly one boundary condition used far more often than any other. It was used by both the BTW and OFC models, so the programme will only consider that one. Other boundary conditions[6] are possible, however, and have been studied elsewhere. BTW and OFC models use very different perturbation techniques to simulate the slow driving force. Although their update rules are very similar, fundamental to the use of cellular automata in general is the wide range of possible rules that simulate different systems. Control mechanisms are very underdeveloped in cellular automata. How exactly this project addresses the topic of control will be discussed later.

Another gripe with existing libraries that aims to be overcome is their narrow breadth of application for what is, in comparison, an uncountable number of research fields. Of course, it would be impossible to satisfy every conceivable complex dynamical system because they are as diverse as nature itself. However, it is worth considering design patterns that may facilitate a wider range of applications than what has already been achieved. This library is novel by way of its extensibility to nearly arbitrary systems.

Finally, a cellular automaton sitting alone in the computer's memory doesn't say much to a prospective researcher. It is necessary to provide a way of extracting data from the system. This is no simple task since the system has many attributes that can be measured on different time-scales and at different points in time along those time-scales. Once the data has been extracted, it would be pertinent to provide built-in plotting functions for the user's convenience. This is mostly missing from the existing libraries. Instead, they provide visualisations of the cellular automaton itself. Feistenauer [9] takes an advanced approach to displaying the array, but this feature is not critical to the project. Since the display of extracted data is only a convenience, only a minor focus is put on its development.

[6]One example is the periodic boundary condition that transforms the usual 2-dimensional plane into a toroidal topology. These have been used to lessen the issue of the finite-size effect

data	parameters	variable methods	constant methods
current grid	seed	initial condition	run
future grid	dimensions	boundary condition	start
mask	transient states	perturbation scheme	mid
series	stable states	update rule	end
	activity setting	control strategy	natural
			manual

Table 3.1: A breakdown of the CellularAutomaton class.

3.1.2 Implementation

The new library is best understood as a collection of classes that serve different purposes. Together, they should provide all the tools required to execute and analyse cellular automata. In fact, the most crucial class to the project is the CellularAutomaton class, while the other two classes facilitate some analysis. There are three additional dataclasses used by a CellularAutomaton object to collect the exported data from the simulations in a formal way.

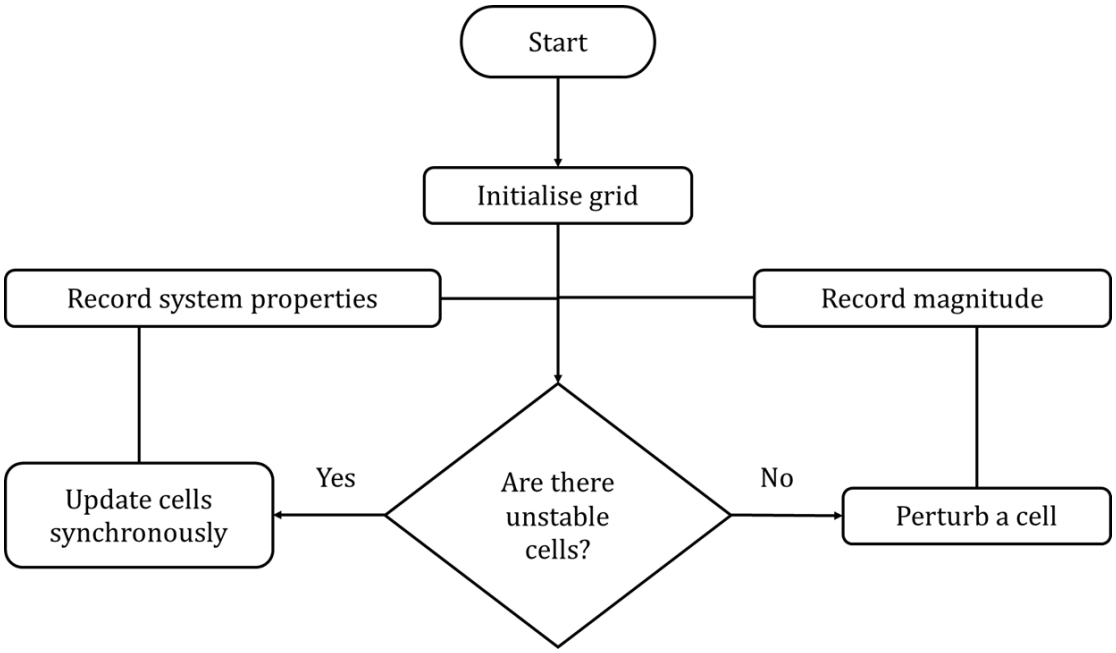

Figure 3.1: A simple flow chart of the slowly driven cellular automaton.

The CellularAutomaton class consists of a large number of attributes that determine the system. Many are variable attributes that can take on different values provided by

the user, while others are internal data structures used to simulate the system and store simulation results. Table 3.1 describes the class. An object of this class can be created by passing a configuration dictionary to the constructor. Parameters and variable methods are defined by the configurations. Upon creating the object, the array is initialised, and methods are defined from among the built-in functions. When the run method is used, see Code E.1 or Figure 3.1, it calls on each of the other constant methods to prepare for the current state's requirements, then searches the present grid for critical cells. These preparations can include perturbing the grid, resetting interim data, or applying "in series" control strategies. It iterates over the set of critical cells and applies the update rule to each. Every call of the update rule returns a set of cells effected by the relaxation, which is joined to the existing set of critical cells. The process keeps iterating over this set, removing one cell each time, until it has been completely exhausted. This marks the end of an event. The update rule does not affect the present grid. Instead, it effects the future grid, which replaces the present grid between time-steps. This ensures the synchronisation of relaxations. The interim data is stored in a Series object via the __record__ method if requested by the user. Flexibility in the storage of data is important for a couple of reasons. First, data may easily be kept in RAM for the duration of the process, but if the user requires copies of arrays between each perturbation, or data between time-steps of an event, the user can quickly run out of RAM. Temporary files were used at one point in development to put large data clusters on the drive, but this was deprecated in favour of more finely tunable output options. Second, recording data takes a non-trivial amount of time once the number of readings becomes large enough. Saving more than is needed by the user can seriously slow down the simulation, so once again, allowing the user to tune the outputs is valuable. Temporary files may be reincorporated eventually, since this output tuning cannot solve the issue of RAM limitations if the user really does intend to save more than their capacity.

The Series dataclass is described approximately in Table 3.2. Each entry can be accessed using dot notation or by a recursive version of the getattr function that is

[state]	**[natural/manual perturb]**	**[natural/manual event]**
	magnitude	
	parallel	[parallel]
	duration	
0	energy	[energy]
	size	[size]
	mass	[mass]
	array	[array]
⋮	⋮	⋮

Table 3.2: A breakdown of the Series dataclass, where [] denotes a list data type.

supplied by the Utilities module. The Utilities module provides various functions, most for internal use while others are for the user, such as the logging functions, which can systematically generate output directories and keep track of executed simulations.

The DataWrangler class, described by Table 3.3, can take data from a Series object and manipulate it into relevant forms for the analysis of complex systems. Currently, it can produce normalised histogram data and perform non-linear regression, which are both very important tools for studying self-organised critical systems. This class also has options for further manipulating the data by averaging over multiple samples, cropping to certain ranges, and outputting the logarithm of the input data. An object should be created by passing the data of interest to the class's constructor, then the wrangle method should be run, which automatically determines the appropriate output form, such as a histogram. The VisualInterface class is mostly a work-in-progress; therefore, it is currently best for the user to use Matplotlib themselves. The plan for this class is to make it easier to generate the more challenging image outputs, such as animated arrays or overlaying multiple results.

generics	**options**	**utility**
data	log data	write data
wrangle	trim up	read data
	trim down	
histograms	**time series**	**optimisation**
hist	series	fitter
avg hist	avg series	basic power law
make hist		modified power law
binner		logged power law

Table 3.3: A breakdown of the DataWrangler class.

3.1.3 User Interface

There are two application programming interfaces (APIs) for the user to engage with the library. The first API is to simply write a config.toml[7] file. This will allow the user to easily select from the range of built-in options for settings like initial conditions, update rules, or control strategies and define their own settings exactly for parameters like dimensions and samples. A config file is included with the library, and it contains default settings. The user should not tamper with these settings but instead write under them to define their own presets. The writing specifications are intuitive, as the user needs only to copy the format of the DEFAULT preset and redefine the settings they wish to change. Multiple presets can be defined one after another by separating each block with a header line containing a unique name. The user can execute the app.py file and will be prompted with a list of presets found in the configuration. Once an option has been selected, the app will parse config.toml to make it suitable for the CellularAutomaton class. The second API requires the user to be more involved with Python and the source code of the project. This way, the user can define entirely new options that can then be written to the configuration or run manually.

[7]The .toml format is a readable and unambiguous configuration specification. It is popular in the Python community. The .ini format was supported in previous versions of this project before Python 3.11 was released, and the code remains archived for backwards compatibility.

keys	defaults	alternative built-in options
samples	2	...
skip transient states	5000	...
desired stable states	5000	...
dimensions	[25,25]	...
initial condition	"rational 0 1"	"rational 10 20", "integer 10 30"
boundary condition	"cliff"	none
perturbation scheme	"global maximise"	"none", "random 1"
update rule	"OFC"	"ABS"
control strategy	"none"	refer to documentation
activity setting	"proactive"	"reactive", "active"

Table 3.4: A breakdown of the configuration file.

3.2 A BTW Implementation

The BTW model, also known as the Abelian Sandpile Model (ASM), was the first model written for this library. In doing so, the BTW model became a built-in model. As stated above, the library aspires in many respects to be as general as possible, so the process of writing any specific model was a test of the library's capabilities. In order to execute a simulation of this model, four functions had to be written. These neatly encompassed four distinct features of the model, so immediately the library appeared approachable in terms of the conceptual abstractions that are required of the user. These four functions define the boundary condition, the initial condition, the perturbation scheme, and the update rule, and are described in more detail in Section 4.

3.3 An OFC Implementation

The OFC model was the second model written for the library. Writing a second built-in model was an instructive experience as it highlighted two benefits of the modular design pattern. First, commonalities between different models can be exploited. Both the BTW model and the OFC model use the same boundary condition. This meant that after writing the corresponding function once, it could be used again without the need for any alterations. This benefit only grows in impact the longer the library exists and

is developed. Second, the process of writing the new functions was well structured since they had to fit a specific signature[8]. Working within a signature focused the effort required and ensured interoperability within the rest of the programme. These both benefit advanced users by reducing the time spent writing their own functions and the time spent debugging if things go wrong.

3.4 The Control Hypothesis

The prospect of control is grand and demanding. It is a highly complex task that may never be fully satisfied, but here an attempt is made at doing so. The challenge arises in a few places. The total dearth of scholarly studies on this specific topic presents the challenge of defining the approach itself. A definition brings many layers of interpretation since it is a legitimate concern to ask about the practicality of control, especially for physical systems governed by natural laws. It is also fair, however, to set aside considerations of practicality, emphasising the universality of self-organised critical systems. The models themselves are less models of specific phenomena than manifestations of fractality. Considering both the practical implications and general applicability of various interpretations of control and the models at large, the following approach has been taken to both facilitate control as part of the library and investigate control as a specific task of the project.

Control of the system has been allocated among several opportunities. The nuanced differences between these opportunities allow for a wide range of interpretations and applications. The concept of time-scales appears once again here. The slow time-scale governs the rate at which a driving force triggers events. The instantaneous time-scale governs the rate at which events unfold. As the names suggest, the instantaneous time-scale allows for events to begin and end instantly from the perspective of the driving force. This is an idealisation in which the driving of the system has no effect on the events themselves other than being their trigger. So, while the driving force is dependent on the

[8] A signature defines the required inputs and outputs of a function.

perturbation scheme function, events are completely dependent on a separate function, the update rule. Control may take place on either time-scale.

Currently, there are four opportunities supported by the library. See Figure 3.2 for how they fit into the simulation process. Two opportunities are governed by the slow time-scale, but differ quite significantly. One is an "in series" perturbation, which takes place immediately after an event finishes. The other is an "in parallel" perturbation that can come in two varieties, periodic and aperiodic. The periodic parallel perturbation specifically measures the driving force magnitude in addition to whatever feedback measurement it uses. The magnitude of the driving force directly correlates to the amount of time passing on the slow time-scale since the rate of driving is assumed to be constant. By measuring this magnitude, control can be activated at regular intervals in time. Alternatively, control can be activated at intervals correlated to the length of time between events. Aperiodic parallel perturbations, as well as series perturbations, use these irregular intervals. The remaining two opportunities are governed by the instantaneous time-scale. Thus, these "concurrent" perturbations can react to the fluctuating, unstable states between the time-steps of individual events. There are two concurrent perturbations since an event can be caused by either the slow driving or the series perturbation.

This project posits that self-organised critical systems can be controlled by feedback series perturbations. The feedback will involve measuring a quantity of the system called mass, which is defined as the mean value of all the cells in the grid. Different control strategies will react depending on this feedback measurement and will be experimentally investigated to determine their effects. Any non-trivial change to the behaviour of the system will be a valuable finding, but in particular, a reduction of large event sizes would be of great significance. The control opportunity chosen is one that makes practical sense in many scenarios. As a reminder, the series perturbation occurs after an event has transpired and the system regains stability, but before the slow driving has any significant

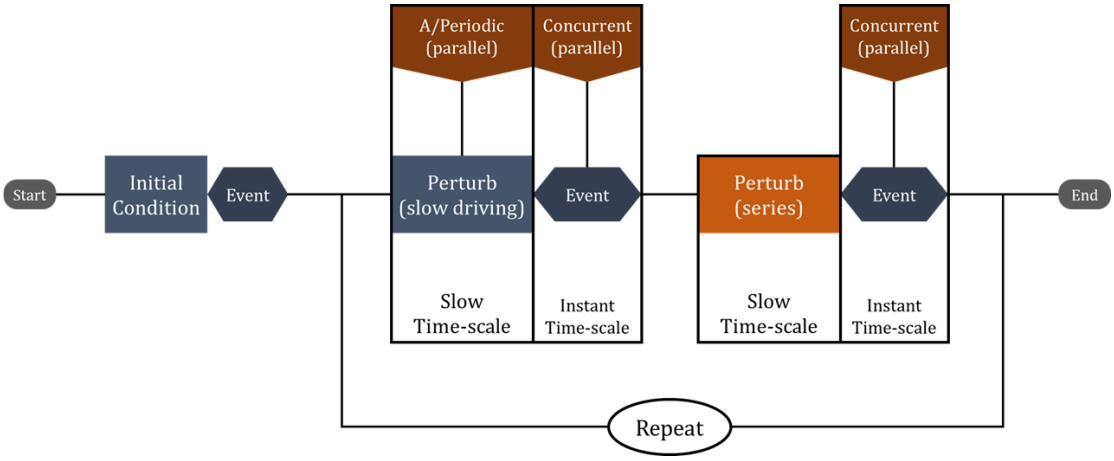

Figure 3.2: A simplified flow chart of the simulation with optional control opportunities coloured orange. The repeat loop iterates however many times as is requested by the user. The event iterations are compiled into the event nodes.

effect on the system. Additionally, this control perturbation only occurs if the feedback measurement meets some criteria defined by the control strategy of interest.

Chapter 4: Measurements and Results

This project encompasses a diverse range of measurements and results that have been extensively studied and evaluated to provide a comprehensive understanding of the subject matter. The findings have been broadly categorised into three distinct sections for ease of comprehension and analysis. The first two sections present the preliminary results that pertain to the quality and accuracy of the novel Python library. The primary focus of this preliminary analysis is to evaluate the library's ability to replicate the results obtained from the BTW and OFC models, created by Bak *et al.* and Olami *et al.* , respectively. These models are widely regarded as exemplary standards for self-organised criticality and have been extensively studied in the research community, both past and present. Therefore, using these models as a baseline, the programme was evaluated to ascertain its reliability and effectiveness in reproducing the original results.

Additionally, since the behaviours of these models are well understood, it becomes easier to assess the impact of the various control strategies on the results, thereby ensuring the validity and credibility of the principal findings. Speaking of which, the third section encompasses the principal results of the experiments conducted using control strategies. These results are of paramount importance as they shed light on the effectiveness of the control measures employed and their impact on the system under study.

4.1 The BTW Model

Undoubtedly, the BTW model [3] is the original and pioneering model in the field of self-organised criticality, and thus it is of vital importance generally speaking. The seminal paper authored by Bak *et al.* was a groundbreaking contribution that revolutionised our understanding of SOC and paved the way for further research in this field. As such, the BTW model occupies a prominent position in this study, and it was the first model to be implemented in the library. A brief, cursory overview of this implementation process is provided in Section 3.2 of this paper.

Moving forward, this section provides a detailed description of the experimental setup used for this preliminary study. This entails a comprehensive discussion of the necessary functions that were written for the CellularAutomaton class and are now included in the built-in repertoire. These functions are designed to facilitate the smooth and efficient execution of the experiments, ensuring that the results obtained are reliable and accurate. These functions cooperate so that the overall simulation is moderately optimised, enhancing the data collection process of the experimental setup. It is possible to investigate the various intricacies of the BTW model in a more nuanced and detailed manner. This section lays the foundation for the subsequent analysis and evaluation of other experimental results.

4.1.1 Description of Options

For an in-depth understanding of the construction of the virtual experimental apparatus, please refer to Section 3.1.2. This section, on the other hand, delves into the essential details of how the BTW model is incorporated into the library. This is done through the functions that represent elements of the decomposed model. All models considered by this project can be decomposed into four distinct types of abstract elements. Those are the initial condition, boundary condition, perturbation scheme, and update rule. These elements are merely aspects that, together, define a model. Each on their own represents features of cellular automaton SOC models in general. The BTW model is defined by a specific element of each of the four types. Although the update rule is the quintessential feature of the BTW model, for the sake of clarity, the BTW model in this project refers to the collection of four specific elements. The library is designed to handle these specific elements by incorporating them into their respective positions in the general flow diagram. This allows for the efficient and seamless execution of the model within the library framework. To this end, this section provides a comprehensive overview of each of these elements and their corresponding functions. By breaking down the model into these abstract elements and corresponding functions, the library provides a powerful and flexible tool for investigating the BTW model and understanding its behaviour.

The grid consists of integer-valued cells. Their actual values depend on the threshold value, which is arbitrarily set to 4. This means that while the system is stable, cells will have values from 0 to 3. As with most dynamical systems possessing an attractor, there is a transient state where the system evolves from some initial condition to its natural state. This may take a few events to reach, but certain initial conditions can hasten the transient period. Bak *et al.* used the initial condition where each cell is given a random value much greater than the threshold. Code 4.1 is the function that generates a grid of any dimension with values between 10 and 30.

Code 4.1: Initial condition of the BTW model.

```
def initial_integer_10_30(self) -> None:
  self.pg = self.rng.integers(10, 30, size=self.dim, endpoint=True)
```

The boundary condition is required whenever the update rule tries to update a cell with coordinates that fall outside the finite grid. When this happens, there needs to be some intervention. The simple boundary condition used for the BTW model is called cliff simply because anything that falls outside the grid disappears from the system. This makes the system as a whole nonconservative, even though the update rule is conservative. Since excess value eventually disappears, events will always stop after enough time. Code 4.2 shows how this is achieved. Whenever a "breach" is detected in the "scope" of an update, "boundary_cliff" is called, and it returns a modified scope that excludes the breach.

Code 4.2: Boundary condition of the BTW and OFC models. Called "cliff" internally.

```
def boundary_cliff(self, breach :tuple[int], magnitude :int|float, scope :set) -> None:
  '''Returns the scope excluding breach cells.'''
  return scope.difference([breach])
```

The perturbation scheme is the slow driving force between events. In this model, that is represented by the addition of 1 to a single cell at random. If this doesn't trigger an event, then a second random cell is perturbed. This repeats until an event is triggered. Code 4.3 shows this function. The while loop repeatedly adds a value of one to a random

cell until it breaks. The magnitude of the perturbation is recorded as the total value added to the system before an event was triggered.

Code 4.3: Perturbation scheme of the BTW model.

```
def perturbation_random_1(self) -> None:
  magnitude = 0
  while True:
      target = tuple(self.rng.integers(self.dim[n]) for n in range(self.ndim))
      self.pg[target] += 1
      self.fg[target] += 1
      magnitude += 1
      if self.pg[target] >= self.threshold: break
  self.data['pert']['magnitude'] += magnitude
  self.control('natural_perturbation_parallel')
```

The update rule is generally considered the defining feature of the model. Part of the ingenuity behind the BTW model is the sheer simplicity of this rule. In two dimensions, a triggered cell redistributes its value to the four orthogonally adjacent cells. These are called its nearest neighbours. The built-in update rule written for the BTW model is generalised to work for any dimensions. The update rule also has the important role of defining the threshold value. In this case, the behaviour of the system is independent of the threshold[9] so a value of 4 is chosen simply because that's the obvious choice for 2-dimensional systems where there are four nearest neighbours. Code 4.4 shows the "rule_BTW" function. Every cell that may trigger an event is passed to this function. If it meets the threshold, then the function constructs the set of nearest neighbours and modifies the grid as needed. It also returns the set of cells that were modified so that they can later be checked themselves for meeting the threshold.

Code 4.4: Update rule of the BTW model. Works for any dimensions.

```
def rule_BTW(self, cell :tuple[int], scale :str, set_threshold :bool) -> set[tuple[int]]:
  if set_threshold: self.threshold = 2 * self.ndim; return
  if self.pg[cell] >= self.threshold:
      self.control(f'{scale}_event_parallel')
      magnitude = 1
```

[9] In other words, the system is not tuned by our choice of threshold.

```
        index_cell = dict(zip([i for i in range(self.ndim)], cell))
        proaction =  {tuple(index_cell[i] for i in index_cell)}
        reaction =   set(tuple(index_cell[i]+1 if n/2 == i
                          else index_cell[i]-1 if n//2 == i
                          else index_cell[i]
                              for i in index_cell)
                          for n in range(self.ndim*2))

        for c in proaction: self.fg[c] -= self.threshold
        for c in copy(reaction):
            if u.dim_check(c, self.dim, self.ndim): self.fg[c] += magnitude
            else: reaction = self.bc(breach=c, magnitude=magnitude, scope=reaction)
        match self.activity:
            case 'proactive': activity = proaction
            case 'reactive':  activity = reaction
            case 'active':    activity = proaction.union(reaction)
        for i in activity: self.mask[i] += 1
        return reaction
    else: return set()
```

4.1.2 Comparison to the Literature

The comparison of results from the simulations to the well-studied results of Bak *et al.* in their original paper on SOC is a crucial verification of the new and yet untested software. It is essential that the results align with those expected from the literature. This would confirm the most rudimentary functionalities of the library or highlight any issues with the project if substantial differences are found. These issues are usually software bugs that have been meticulously eliminated since being uncovered so that the results shown here represent the library at its best. Many errors and anomalies were encountered, but all those that prevented the simple completion and measurement of a successful simulation were addressed.

Figure 4.1 is reproduced from a paper authored by Bak, Tang, and Wiesenfeld and published in Physical Review A in 1988. They studied systems they built from scratch for SOC in one and two dimensions. This figure shows their results from simulating the BTW model in accordance with the rules given by Equations 4.1 and 4.2. Section 2.1

describes the original work of Bak *et al.* in more detail.

$$z_t(x, y) \rightarrow z_{t-1}(x, y) - 4 \tag{4.1}$$

$$z_t(x \pm 1, y \pm 1) \rightarrow z_{t-1}(x \pm 1, y \pm 1) + 1 \tag{4.2}$$

Between events, the system is driven on a slow time-scale by adding a value of one to a random cell. This may or may not trigger an event, so repeated perturbations may be required. Equation 4.3 describes this perturbation scheme.

$$z_t(x, y) \rightarrow z_{t-1}(x, y) + 1 \tag{4.3}$$

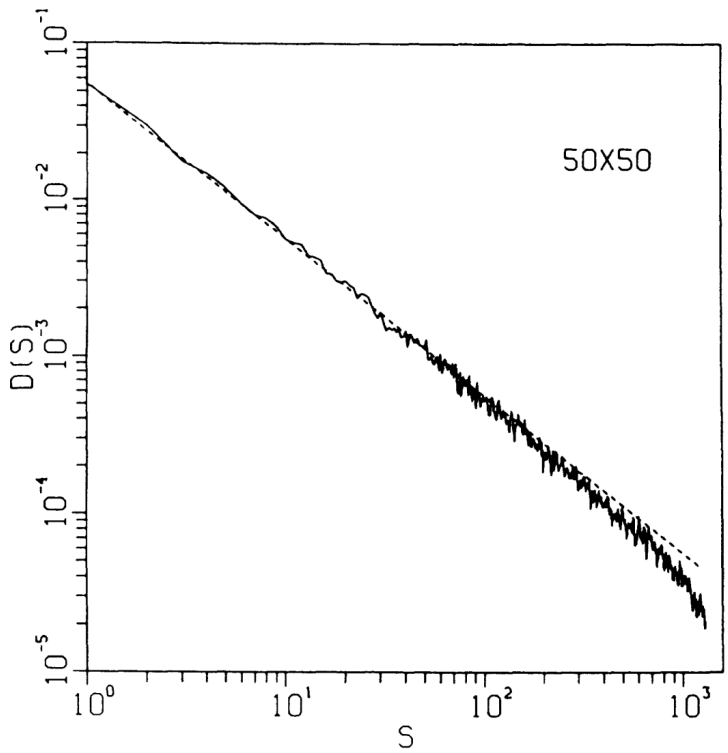

Figure 4.1: Reproduced from Bak *et al.* [4]. Cluster size distribution for a 50×50 array and 100000 perturbations were performed.

Figure 4.1 exemplifies the power-law relationship typical of self-organised criticality and fractality in general. This power-law also represents the scale-invariance of the system, which is fundamental to criticality. It is the primary tool used in this project to analyse results, and numerical findings will be discussed in Section 5. For now,

observe the uniformity of the Bak's results on the straight, dotted line. Noise creeps into the measurements as the sizes increase. The trend also slightly deviates from the straight line at the absolute largest sizes. To acquire these measurements, Bak studied a two-dimensional, 50×50 array. He performed 100000 perturbations, Eq. 4.3, to cause enough events for this figure. This generated a smaller number of events, or readings.

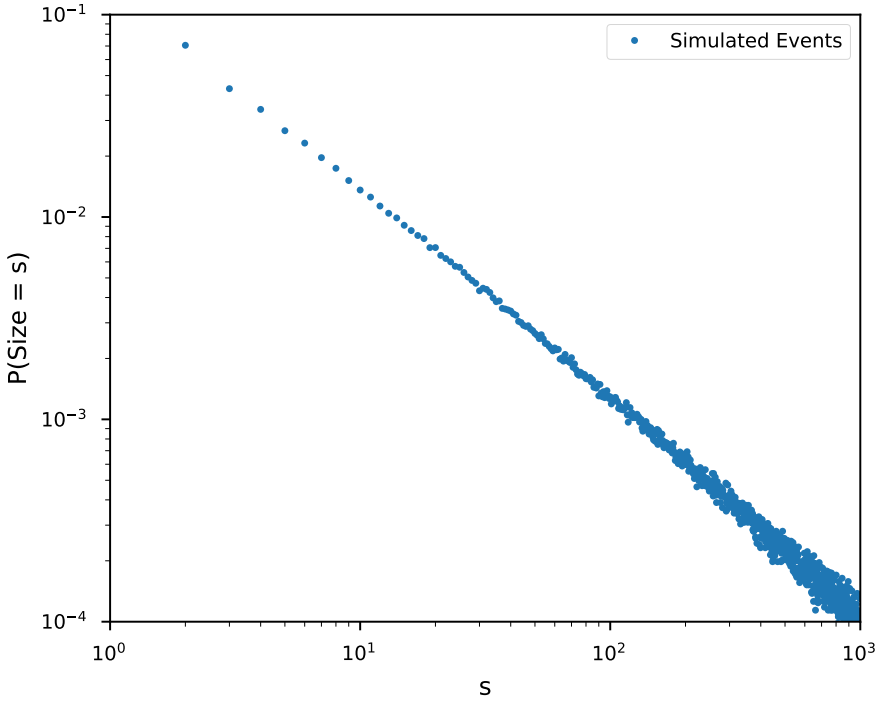

Figure 4.2: Event size distribution of the BTW model simulated using the new library. The grid dimensions were 50×50 and 10000 events were observed.

Now contrast the original findings with those of Figure 4.2. This was generated using the new software library in 10 minutes. Similarly, a 50×50 array was studied, but only 10,000 events were measured. To reduce the noise in those readings, the simulation was repeated 10 times using the same settings but with different initial grids. The figure shows the average of these 10 samples. This clarifies the trend in the data, particularly at large sizes. These results do not exactly match those of Bak, as they appear to have been translated. However, this is not a difference in the system's behaviour but rather a minor discrepancy in the processing of data. The ordinate data is probability normalised to the number of readings, so since Bak took more readings, the numbers are shifted. Importantly, this does not change the power-law in any significant way.

4.2 The OFC Model

4.2.1 Description of Options

The OFC model [18] is in some ways different from the BTW model, while in other ways it is the same. Although the literature often sees these two separately, it is worth considering that the OFC model is partially a generalisation of the BTW model. The abelian sandpile model is more often cited to this end, but there are many similarities with the OFC model that should not be overlooked. Owing to the universality of fractals, these models behave very similarly, despite the OFC model being heralded as realistic and used to explicitly model earthquakes while the BTW model has remained generic.

One big difference between these models is the initial condition. While the BTW model dealt with discrete integer arithmetic, the OFC model extends this to rational numbers, or floating-point numbers. Also, the grid is populated with values near the threshold rather than much greater than those. There is an interesting discussion on the effects of these two approaches to initial conditions. For now, the transient state is hopefully avoided by providing ample time for the system to evolve before taking measurements. Once again, the threshold is arbitrary, so a value of one is chosen for simplicity. See Code 4.5 for this initial condition's built-in function.

Code 4.5: Initial condition of the OFC model.

```
def initial_rational_0_1(self) -> None:
  self.pg = self.rng.random(size=self.dim, dtype=np.float64)
```

The boundary condition is identical to that of the BTW model. While it is true Bak *et al.* investigated a few variations of the "cliff" condition, this one alone was chosen to implement both models for the sake of cohesion. Olami *et al.* used solely the cliff condition. Refer to Code 4.2 from earlier.

The perturbation scheme is the slow driving force between events. In this model,

that is represented by the addition of 1 to a single cell at random. If this doesn't trigger an event, then a second random cell is perturbed. This repeats until an event is triggered. Code 4.3 shows this function. The while loop repeatedly adds a value of one to a random cell until it breaks. The magnitude of the perturbation is recorded as the total value added to the system before an event was triggered.

The perturbation scheme is significantly different. The BTW model uses "local" driving. This meant that one cell was perturbed between events. The OFC model uses "global" driving. This means that every cell is perturbed by the same amount between events. The magnitude of the perturbation is determined by taking the difference between the highest value in the grid and the threshold value. This guarantees that an event is triggered. Code 4.6 demonstrates how this appears even simpler than the BTW model's local driving.

Code 4.6: Perturbation scheme of the OFC model.

```
def perturbation_global_maximise(self) -> None:
  magnitude = self.threshold - np.amax(self.pg)
  self.data['pert']['magnitude'] += magnitude
  self.pg += magnitude
  self.control('natural_perturbation_parallel')
```

Just as before, the update rule is the central aspect of the model. The "rule_OFC" function is very similar to the "rule_BTW" function, as can be seen in Code 4.7. There are, however, a couple of differences. The first is that instead of subtracting the threshold from a triggered cell, its value is set to zero. The second is that there is a conservation factor that defines the percentage of a cell's value that gets redistributed. A conservation factor of 1 would be equivalent to the fully conservative BTW model, although Olami *et al.* primarily studied the case where the conservation factor was 0.8. These differences can be explained by putting the system in the context of earthquake modelling, as discussed more in Section 2.2. Setting a cell to zero resembles the stick-slip motion of spring-block models or tectonic plates, and the conservation factor is an effort to more

closely represent nonconservative natural phenomena.

Code 4.7: Update rule of the OFC model. Works for any dimensions.

```
def rule_OFC(self, cell :tuple[int], scale :str, set_threshold :bool) -> set[tuple[int]]:
  if set_threshold: self.threshold = 1; return
  if self.pg[cell] >= self.threshold:
    self.control(f'{scale}_event_parallel')
    conservation = 0.8 # factor divided by 2 * ndim to determine dissipative effect
    magnitude = self.pg[cell] * (conservation/(self.ndim*2))
    index_cell = dict(zip([i for i in range(self.ndim)], cell))
    proaction =  {tuple(index_cell[i] for i in index_cell)}
    reaction =   set(tuple(index_cell[i]+1 if n/2 == i
                      else index_cell[i]-1 if n//2 == i
                      else index_cell[i]
                          for i in index_cell)
                      for n in range(self.ndim*2))
    for c in proaction: self.fg[c] = 0
    for c in copy(reaction):
      if u.dim_check(c, self.dim, self.ndim):
        self.fg[c] += magnitude
      else: reaction = self.bc(breach=c, magnitude=magnitude, scope=reaction)
    match self.activity:
      case 'proactive': activity = proaction
      case 'reactive':  activity = reaction
      case 'active':    activity = proaction.union(reaction)
    for i in activity: self.mask[i] += 1
    return reaction
  else: return set()
```

4.2.2 Comparison to the Literature

Figure 4.3 is reproduced from the 1992 paper by Zeev Olami, Hans Jacob S. Feder, and Kim Christensen [18], published in Physical Review Letters. The authors were concerned with the dynamics of earthquakes and decided that the relatively new concept of self-organised criticality could shed some light on the topic. At the time, most research into this mechanism focused on conservative cellular automata. The OFC model changed this precedent with its conservation factor. Whether or not this factor constitutes tuning the system or not is debated, but Olami *et al.* were clear that the intent was not to manufacture fractal behaviour but rather to better model real-world

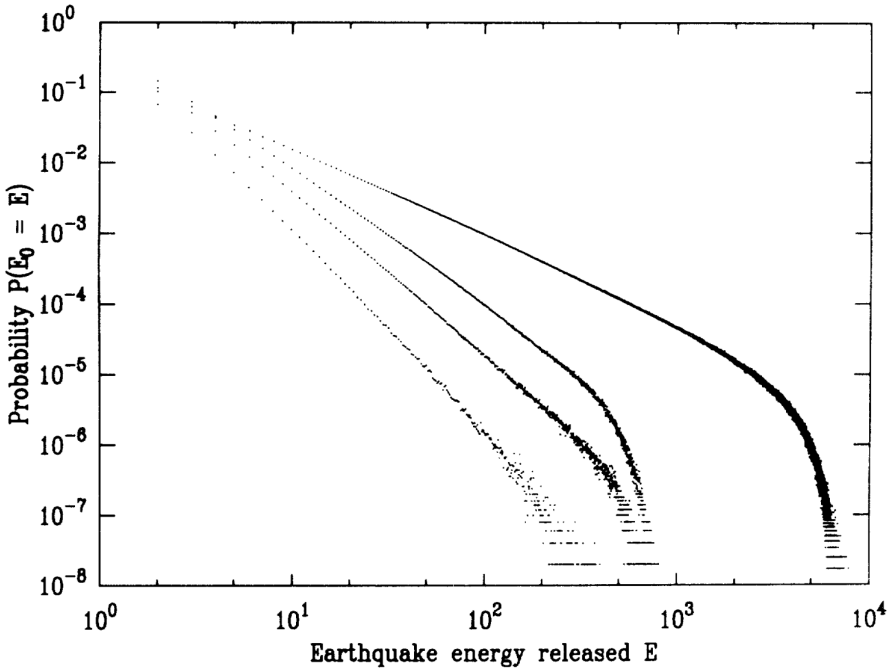

Figure 4.3: Reproduced from Olami *et al.* [18]. Earthquake size distributions for a 35×35 array with the following levels of conservation: 100%, 80%, 60%, and 40%, from top to bottom.

measurements of earthquake systems and their levels of conservation.

Figure 4.3 and Figure 4.4 show the probability distributions of simulated earthquakes for different levels of conservation. Olami *et al.* measured earthquake energies, which are directly proportional to their sizes. Again, despite the differences at extreme sizes, the correlation between these results is brilliant. Those extreme size differences are due to the fact that the exact specifications of Olami's simulations are unknown. For instance, Olami stated that they used 35×35 arrays for the figure above, but their 100% conservation curve clearly proves that was not the case since its cut-off is nearly an order of magnitude greater than the size of the array it was supposedly simulated with. These discrepancies, however, do not detract from the message that these systems exhibit key self-organised critical features.

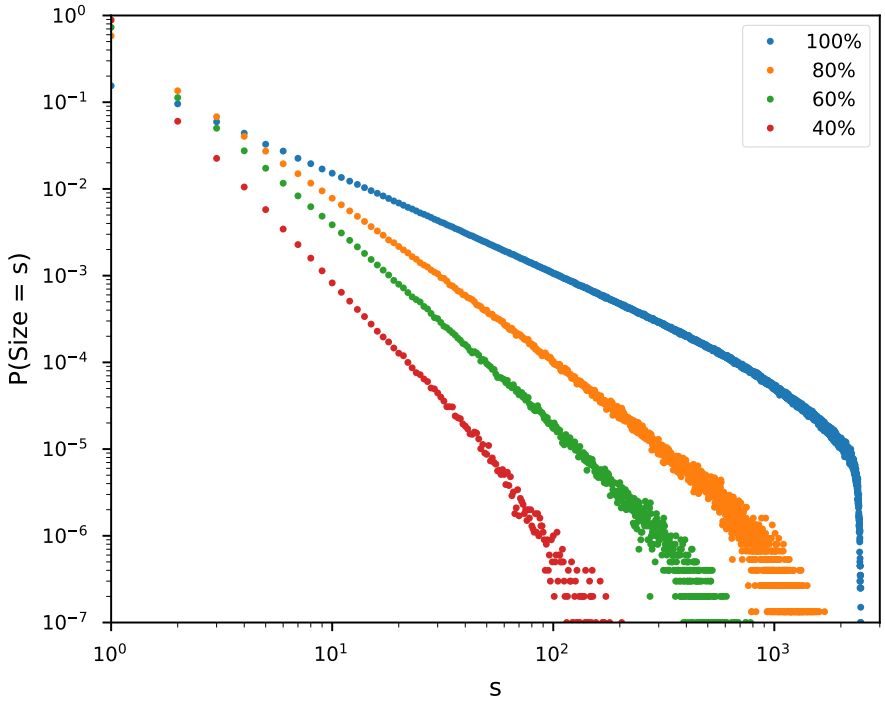

Figure 4.4: Event size distribution of the OFC model for various levels of conservation, averaged over 20 samples. The grid dimensions were 35×35 and 500000 events were observed.

4.3 Controlled Experiments

4.3.1 Strategies

Two control strategies have been devised for application to the OFC model. Since control has not been studied in this context before, the process was exploratory. Ultimately, two promising strategies will be showcased in the following sections, each with a number of variations. The primary goal kept in mind while developing this strategy was the reduction of large event sizes. This will be discussed in Section 6. Crucial to both strategies is the mass of the system after events. The mass is a quantity defined here as the average value of all the cells in the grid. This quantity may have some connection to the behaviour of the system, and the control results suggest this is so. The instantaneous mass is the mass measured at a single point in time. The reference mass is a quantity determined empirically by analysing the typical mass of the uncontrolled system. Averaging the masses of the OFC model after many events yields a certain value with an extremely low standard deviation. This value is used as a reference mass

by both strategies.

The first is the linear differential control strategy. This algorithm takes a measurement of the instantaneous mass and finds the difference between that and the reference mass. Only if the instantaneous mass is less than the reference mass will it perturb the system. If this criteria is met, the perturbation applies to one random cell in the grid with a magnitude of the absolute value of the difference. Variations on this strategy involve amplifying the perturbation by a range of values.

The second is the fixed impulse control strategy. This algorithm likewise takes a measurement of the instantaneous mass. Depending on this feedback measurement, the system may be perturbed by a constant value, 1, at a random cell. The two variations of this strategy differ in when they decide to perturb. One acts when the instantaneous mass is less than the reference mass, as in the first strategy, while the other acts when the feedback is greater than the reference. The "lesser" variant is similar to the first strategy, but the constant perturbation magnitude highlights the causality of these strategies.

4.3.2 Measurements of Control Effects

Here are the measurements for both strategies. Figure 4.5 shows the results for the first one, the linear differential control strategy. The uncontrolled system is compared to three variants of control. Δ is the difference between the instantaneous mass and the reference mass. It is then amplified by multiples of 2.

The second strategy, the fixed impulse control strategy, reveals some interesting changes in behaviour. Figure 4.6 shows the two variants compared to the uncontrolled system. M_i and M_r refer to the instantaneous and reference masses, respectively. The lesser variant looks similar to the results from Figure 4.5, but exacerbated. The greater variant takes a big departure from what is seen in the other strategies.

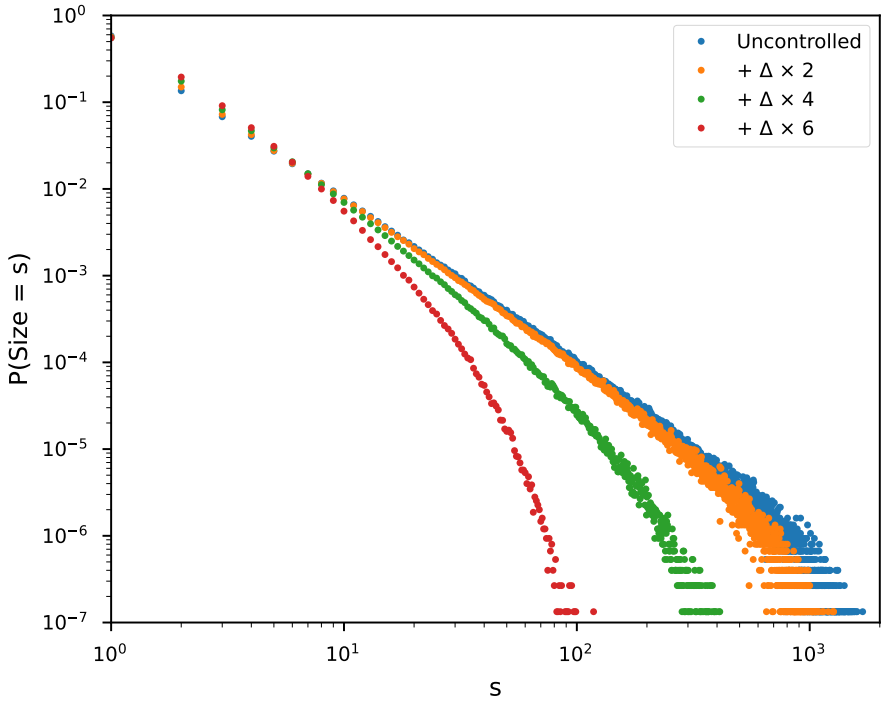

Figure 4.5: Event size distributions of the controlled OFC model, averaged over 15 samples. The grid dimensions were 50×50 and 500000 events were observed. The system is influenced by three variants of the linear differential control strategy.

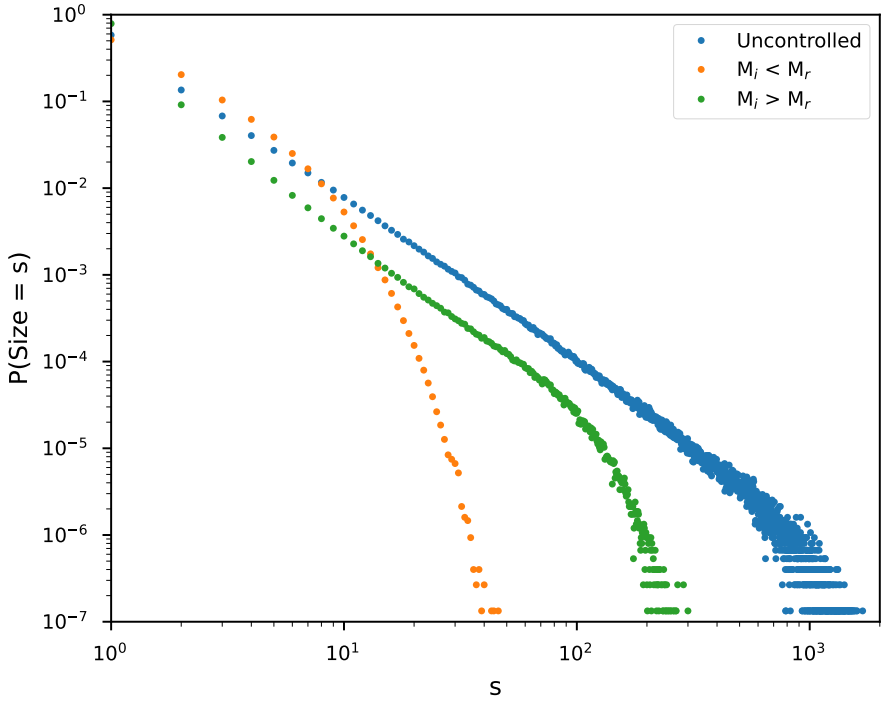

Figure 4.6: Event size distributions of the controlled OFC model, averaged over 15 samples. The grid dimensions were 50×50 and 500000 events were observed. The system is influenced by the two variants of the fixed impulse control strategy.

Chapter 5: Analysis

The exponent of the power-law, α, is the most indicative parameter of the system's behaviour in the context of searching for SOC. The simplest way to determine the exponent of data that follows this relationship is to plot the data on log-log axes and fit a straight line. Equation 5.2 shows how this can easily produce the exponent. Only the slope of the linear function needs to be determined.

$$P(s) \sim s^{-\alpha} \tag{5.1}$$

$$\log(P(s)) = -\alpha \log(s) + C \tag{5.2}$$

An alternative relationship, Eq. 5.3, has been investigated that attempts to correct the error induced by the cut-off at large event sizes. The introduction of an exponential decay term hopefully models the deviation from a straight line. The successes of this relationship are interesting.

$$P(s) \sim s^{-\alpha} e^{-s/l} \tag{5.3}$$

In this relationship, l is the scaling factor that correlates to the dimensions of the grid. The exponent, s, is still the decisive parameter that should be determined. Analysing the data with this relationship is more difficult, however, since non-linear regression techniques are required. To distinguish between the two possible relationships, Equation 5.1 will be called the ordinary power-law, and Equation 5.3 will be called the modified power-law.

5.1 Optimising the BTW Model

Bak *et al.* found exponents "tolerably close to one (and certainly between 0 and 2)". The new results show that the library indeed simulates the BTW model successfully. This was established early in the development process and provided the framework with which other models could be simulated and analysed. Using the ordinary power-law, an

exponent of 1.018 was found. Using the modified power-law, an exponent of 0.978 was found. These align with the expectations superbly. So far, both relationships perform well, although the modified power-law doesn't quite mimic the cut-off exactly. Figures 5.1 and 5.2 demonstrate these findings.

5.2 Optimising the OFC Model

There is much to analyse in the OFC model. The four levels of conservation are fitted to both the ordinary power-law Figure 5.3, and modified power-law, Figure 5.4, relationships. Olami *et al.* do not explicitly state what values they calculated for these curves, but a rough estimate can be extrapolated from their paper. Using the ordinary power-law, the exponents for 100%, 80%, 60%, and 40% conservation are 0.169, 0.856, 1.258, and 1.883. Using the modified power-law, the exponents are found to be 0.798, 0.825, 1.245, and 1.971. On the whole, these numbers are good matches to the original paper, especially the ordinary power-law. Here, the modified power-law produces tremendous fits for 80% and 60% conservation, slightly worse for 40%, and terribly for 100%. The question of whether or not the modified power-law is a suitable relationship has no clear answer yet.

5.3 Analysis of Control Effects

The analysis of the control strategies and their effects is not as simple as fitting the data to a function since the effect of the control is to disrupt the behaviour of the system. Thus, they almost certainly will not follow the ordinary power-law relationship. The visual representation of the data in a graph provides ample insights, however. Dramatic changes in behaviour are obvious, so their precise modelling may come later.

Almost surprisingly, the modified power-law relationship can fit some of the results remarkably well. Figure 5.5 is the first evidence of that with an application of the relationship to the first strategy, the linear differential control strategy. Unless the magnitude of the control strategy's perturbation is relatively small, the controlled curves

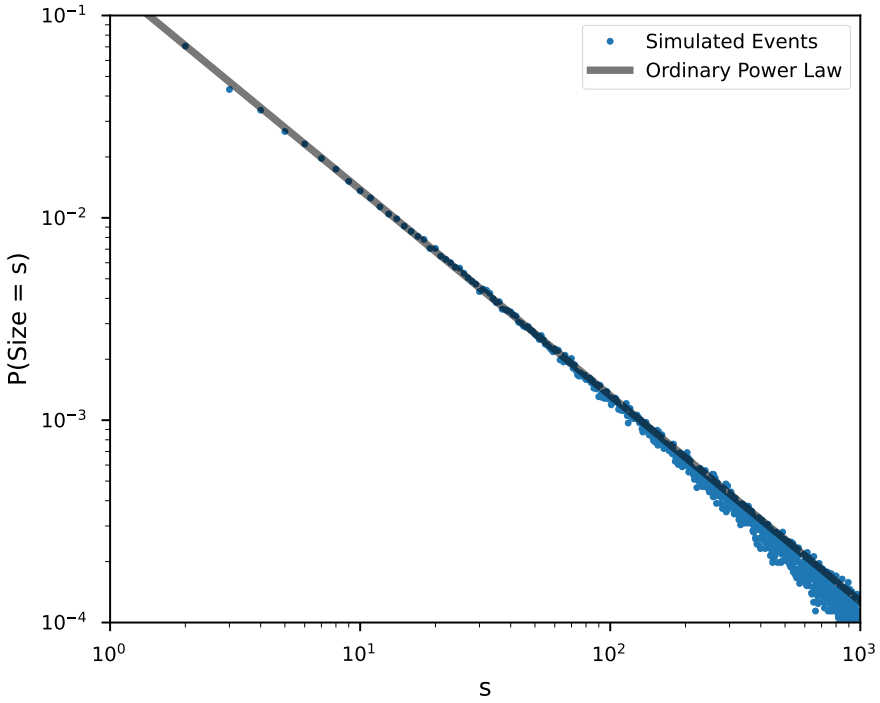

Figure 5.1: Event size distribution of the BTW model, averaged over 10 samples. The grid dimensions were 50×50 and 50000 events were observed. The measurements are fitted to the ordinary power-law relationship with an exponent of 1.018.

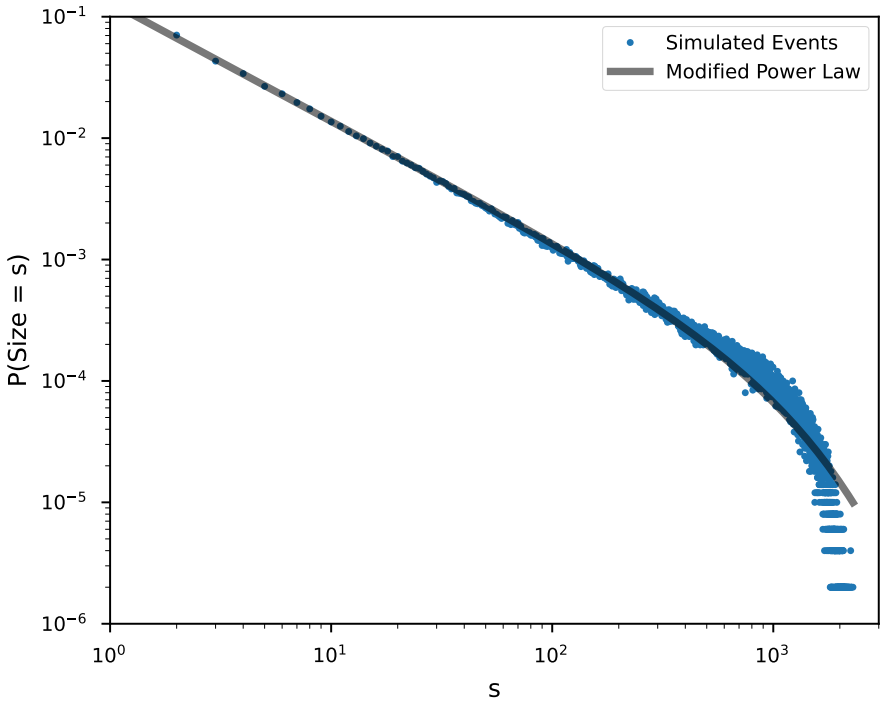

Figure 5.2: Event size distribution of the BTW model, averaged over 10 samples. The grid dimensions were 50×50 and 50000 events were observed. The measurements are fitted to the modified power-law relationship with an exponent of 0.978.

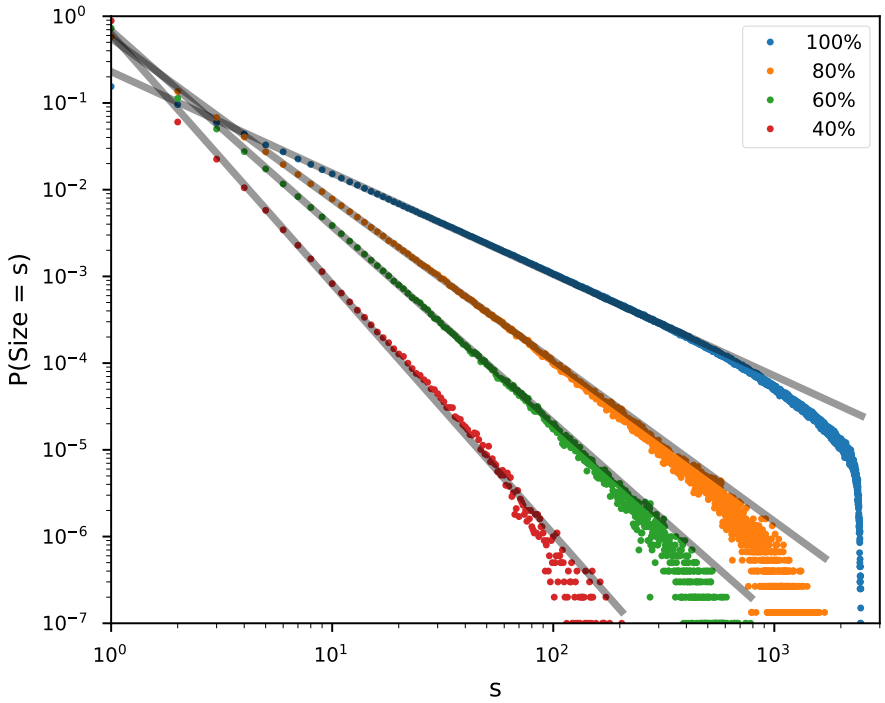

Figure 5.3: Event size distributions of the OFC model for various levels of conservation, averaged over 20 samples. The grid dimensions were 35×35 and 500000 events were observed. The measurements are fitted to the ordinary power-law relationship with exponents: 0.169, 0.856, 1.258, and 1.883.

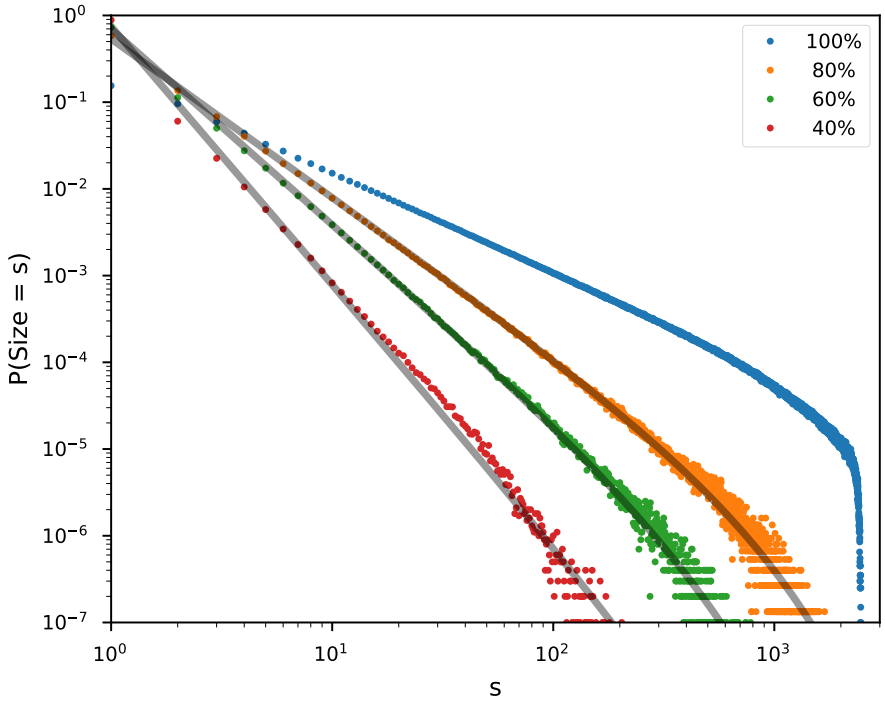

Figure 5.4: Event size distributions of the OFC model for various levels of conservation, averaged over 20 samples. The grid dimensions were 35×35 and 500000 events were observed. The measurements are fitted to the modified power-law relationship with exponents: 0.798, 0.825, 1.245, and 1.971.

do not fit a straight line like the uncontrolled curve does. In fact, the exponential decay of the modified power-law exactly models the reduction in the frequency of large events. Since the magnitude of the control increase is linear across the variants, this strategy exhibits a very effective influence. Unfortunately, the success of the modified power-law in fitting these curves does not yield significant numerical results currently. The exponents don't insinuate the changes. They are 0.842, 0.853, 0.882, and 0.614, reading from the uncontrolled curve to the 6-times-amplified control variant. The real changes come from the scaling factors l, which are 735, 451, 73, and 11. Although the correct interpretation of these is unclear.

Figure 5.6 tries to take the same approach with the second strategy, the fixed impulse control strategy. Unsurprisingly, the modified power-law reveals nothing new about the greater variant, which appears to still follow an ordinary power-law relationship similar to the uncontrolled curve. The lesser variant is perfectly modelled by the modified power-law however. Evidently, the events of the lesser variant system primarily follow an exponential decay distribution.

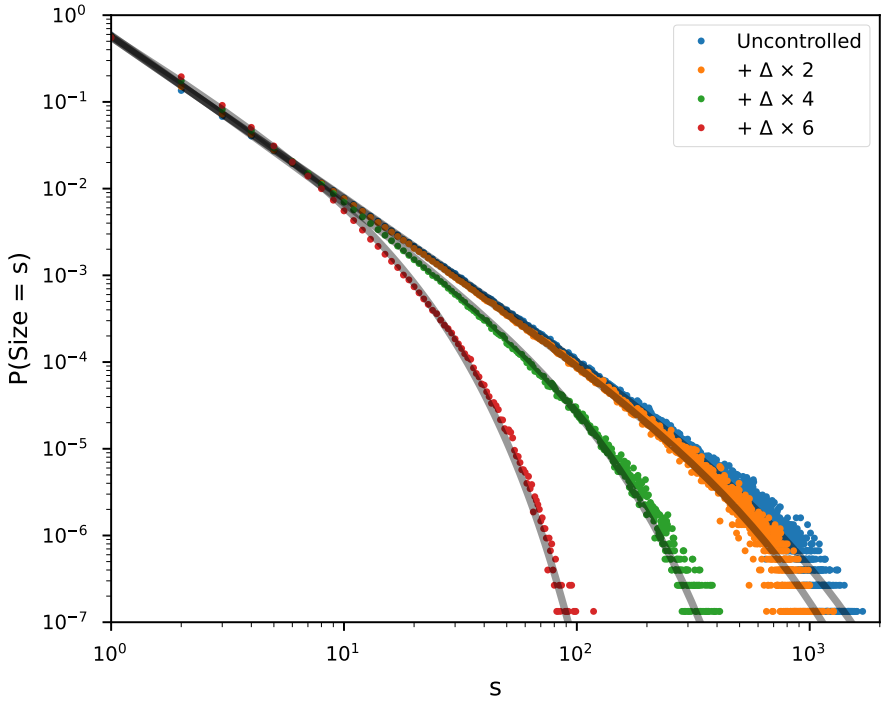

Figure 5.5: Event size distributions of the controlled OFC model, averaged over 15 samples. The grid dimensions were 50×50 and 500000 events were observed. The system is influenced by three variants of the linear differential control strategy. The measurements are fitted to the modified power-law relationship with exponents: 0.842, 0.853, 0.882, and 0.614.

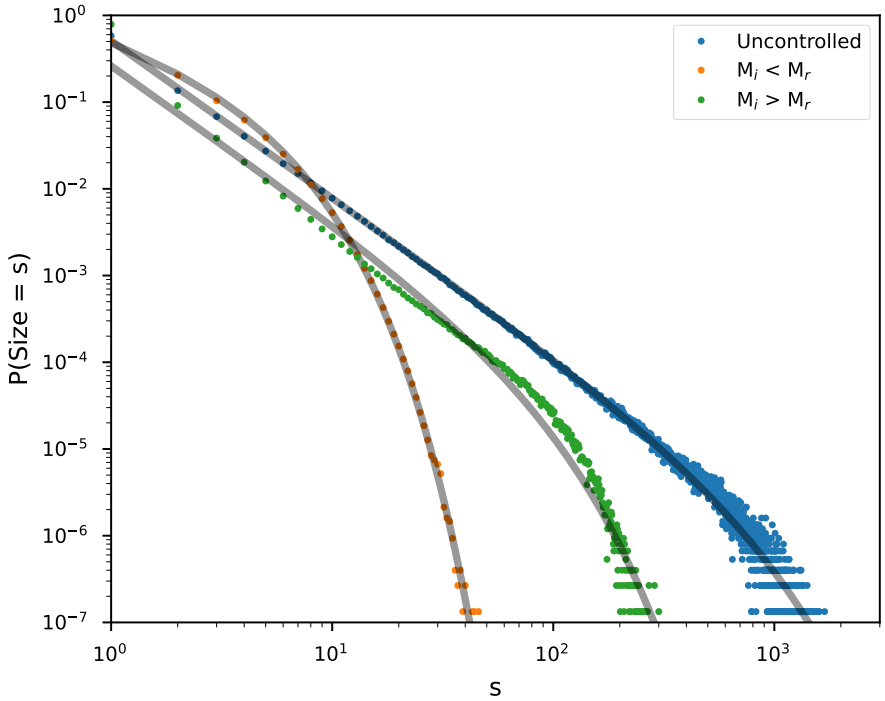

Figure 5.6: Event size distributions of the controlled OFC model, averaged over 15 samples. The grid dimensions were 50×50 and 500000 events were observed. The system is influenced by the two variants of the fixed impulse control strategy. The measurements are fitted to the modified power-law relationship with exponents: 0.809, -0.233, and 0.789.

Chapter 6: Interpretation and Discussion

In this section, an interpretation and discussion of the results are made to emphasise the importance of the topic and hopefully clarify certain troubles. The results themselves are detailed in Sections 4 and 5. This discussion should call attention to the context of the literature review. Some potential questions that might arise from the findings of this project will be addressed in light of any implications the analysis might have suggested. This work extends beyond what is known in the literature, so the general approach may be judged and surely improved.

In this project, a novel Python library for cellular automaton simulations is developed. After a critical appraisal of the myriad existing programmes for both Python and MATLAB, it became clear that their functionalities were insufficient. Each had their own merits, however, and they inspired the ambition and scope of the new library. Flexibility and extensibility are the hallmarks of the new library. The ease with which a user can adjust the programme to simulate a wide range of models was exemplified when the BTW and OFC models were implemented. The library also includes analysis tools and control opportunities that allow users to conveniently analyse and control the behaviour of the system. The control functionality is unique to this library and required much innovation to integrate properly.

This project focused on these two models in particular since they are widely recognised as self-organised critical systems and possibly the most well established in the literature. These were important factors in the decision-making process since they facilitated the aims of the project. First, the Python library could easily be verified against the existing, well-tested, and oft-reproduced results of Bak and Olami. Second, the effects of the control strategies wouldn't be conflated with the system's ordinary behaviour.

The analysis of the statistical properties of the BTW and OFC models revealed that they exhibit power-law behaviour, consistent with previous studies. In the case of the OFC model, it was shown that the exponent of the power-law was dependent on the level of conservation in the system. It has been touted that the level of conservation is a tuning parameter and therefore the OFC model doesn't meet the criteria for SOC; however, this misinterprets the reason for having a nonconservative system. Which is to model physical systems such as seismicity.

The results showed that the control strategies successfully influenced the behaviour of the OFC model, most notably by reducing the frequency of large events. This finding is significant because it demonstrates the potential of control strategies to create desirable properties in SOC systems. One such property must be the reduction of extreme events such as earthquakes, avalanches, or market crashes. In systems such as these, big spells disaster. This project adds to a growing body of literature that has explored the effectiveness of control strategies in various complex systems, such as financial markets, climate dynamics, and neural networks. What is special about the work of this project is that these systems are self-organised, which has interesting implications for the underlying mechanisms that govern the natural world.

The results of a series of control experiments suggest that the modified power-law is indicative of the effects of the induced dynamics from control strategies. Seeing the reduced frequency of events may prompt the question of where those events go. It is reasonable to fear that the manual perturbation of a control strategy may itself cause large events. Figure 6.1 takes the linear differential control strategy and clearly alleviates this concern. The manual events from the control are substantially infrequent.

The project has some weaknesses, however, that should be considered. First, the control strategies were devised through a heuristic approach, and it is unclear whether they would work for other cellular automata models or under different conditions. Future

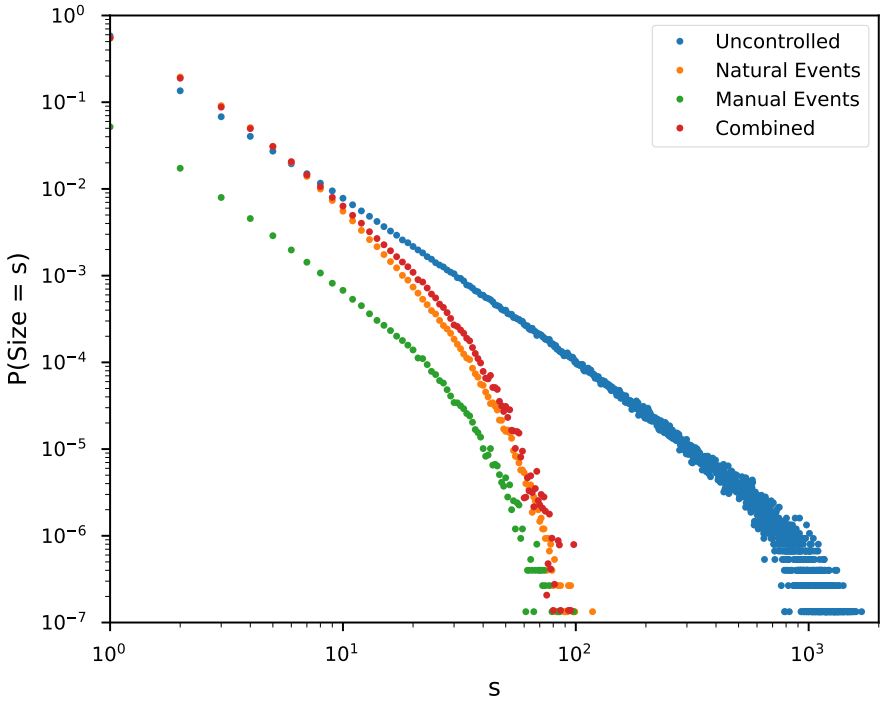

Figure 6.1: Event size distributions of the controlled OFC model, averaged over 15 samples. The grid dimensions were 50×50 and 500000 events were observed. The system is influenced by a variant of the linear differential control strategy. Shown are events caused by the natural slow driving as well as those caused manually by the control perturbation itself.

work could explore control strategies with a more rigorous mathematical perspective. Then they may be applicable to a wider range of models, and the actual dynamics of those strategies may be better understood. Secondly, the analysis techniques used were limited to the statistical properties of the system and could only provide surface-level insights into the underlying dynamics of the system. More advanced analysis techniques, such as mean-field theory, should be utilised. Despite these demerits, this work demonstrates the effectiveness of the new Python library for simulation, analysis, and control of cellular automata. This hopefully promotes the burgeoning field of complex systems to grow further.

Chapter 7: Conclusions

The first contribution of this project was the development of a novel Python library for cellular automaton simulations that improves upon the existing options for either Python or MATLAB. The library provides sufficient tools to analyse the statistical properties of sel-organised critical systems and also to visualise those analyses and the associated array itself. The greatest feature of the library, however, is its extensibility. The user may implement diverse models of their own accord, while the BTW and OFC models are built-in. The hope is that this library will be a valuable tool for researchers interested in studying complex systems using cellular automata.

The second contribution of this project was the investigation of two control strategies on the OFC model using the newly developed library. It was shown that the control strategies successfully influenced the behaviour of the system in a non-trivial capacity. In particular, the frequency of large events was able to be reduced, which is a highly desirable change of properties in many applications. Moreover, the ability to control the behaviour of these systems opens up many new possibilities for research and practical applications.

References

[1] Luis M Antunes. "CellPyLib: A Python Library for working with Cellular Automata". In: *Journal of Open Source Software* 6 (67 2021), p. 3608. DOI: 10.21105/joss.03608

[2] Karl Johan Astrom and Richard M. Murray. *Feedback Systems : An Introduction for Scientists and Engineers*. 2nd ed. Princeton University Press, 2021. ISBN: 9780691193984

[3] Per Bak, Chao Tang, and Kurt Wiesenfeld. "Self-organized criticality: An explanation of the 1/f noise". In: *Phys. Rev. Lett.* 59 (1987), p. 381. DOI: 10.1103/PhysRevLett.59.381

[4] Per Bak, Chao Tang, and Kurt Wiesenfeld. "Self-organized criticality". In: *Phys. Rev. A* 38 (1 1988), pp. 364–374. DOI: 10.1103/PhysRevA.38.364. URL: https://link.aps.org/doi/10.1103/PhysRevA.38.364

[5] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, 1999. ISBN: 9780195131581. DOI: 10.1093/OSO/9780195131581.001.0001

[6] Robert Burridge and Leon Knopoff. "Model and theoretical seismicity". In: *Bulletin of the Seismological Society of America* 57 (1967), pp. 341–371. DOI: 10.1785/BSSA0570030341

[7] H. Dashti-Naserabadi and M. N. Najafi. "Bak-Tang-Wiesenfeld model in the upper critical dimension: Induced criticality in lower-dimensional subsystems". In: *Physical Review E* 96 (4 Oct. 2017), p. 042115. ISSN: 2470-0045. DOI: 10.1103/PhysRevE.96.042115

[8] Deepak Dhar. "The abelian sandpile and related models". In: *Physica A* 263 (1999), pp. 4–25. DOI: 10.1016/S0378-4371(98)00493-2

[9] Richard Feistenauer. *cellular-automaton PyPi*. 2021. URL: https://pypi.org/project/cellular-automaton/

[10] Andrew Fowler and Mark McGuinness. *Chaos, An Introduction for Applied Mathematics*. Springer Nature Switzerland AG, 2019. ISBN: 978-3-030-32537-4. DOI: 10.1007/978-3-030-32538-1

[11] João P. Hespanha. *Linear Systems Theory*. 2nd ed. Princeton University Press, 2018

[12] John H. Holland. *Complexity, A Very Short Introduction*. Oxford University Press, 2014. ISBN: 9780199662548

[13] Henrik Jeldtoft Jensen, Kim Christensen, and Hans C. Fogedby. "1/f noise, distribution of lifetimes, and a pile of sand". In: *Physical Review B* 40 (10 Oct. 1989), p. 7425. ISSN: 01631829. DOI: 10.1103/PhysRevB.40.7425

[14] Benoit B Mandelbrot and John W Van Ness. "Fractional Brownian Motions, Fractional Noises and Applications". In: *SIAM Review* 10 (1968), pp. 422–437. DOI: 10.1137/1010093

[15] Benoit B. Mandelbrot. *The Fractal Geometry of Nature*. 1977. ISBN: 9780716711865

[16] Benoit B. Mandelbrot. "Fractal Aspects of the Iteration of z -¿ Lambda z(1 - z) for Complex Lambda and z". In: *Annals of the New York Academy of Sciences* 357 (1 Dec. 1980), pp. 249–259. ISSN: 1749-6632. DOI: 10.1111/J.1749-6632.1980.TB29690.X

[17] Sergei Maslov, Chao Tang, and Yi Cheng Zhang. "1/f Noise in Bak-Tang-Wiesenfeld Models on Narrow Stripes". In: *Physical Review Letters* 83 (12 Sept. 1999), p. 2449. ISSN: 10797114. DOI: 10.1103/PhysRevLett.83.2449. URL: https://journals.aps.org/prl/abstract/10.1103/PhysRevLett.83.2449

[18] Zeev Olami, Hans Jacob S Feder, and Kim Christensen. "Self-organized criticality in a continuous, nonconservative cellular automaton modeling earthquakes". In: *Phys. Rev. Lett.* 68 (1992), p. 1244. DOI: 10.1103/PhysRevLett.68.1244

[19] Dietmar Plenz et al. "Self-organized criticality in the brain". In: *Frontiers in Physics* 9 (2021), p. 639389. DOI: 10.3389/fphy.2021.639389

[20] William H Press. "Flicker noises in astronomy and elsewhere". In: *Comments on Modern Physics* 7 (1978), pp. 103–119

[21] Alexander Shapoval, Boris Shapoval, and Mikhail Shnirman. "1/x power-law in a close proximity of the Bak-Tang-Wiesenfeld sandpile". In: *Scientific Reports* 11 (1 Sept. 2021), p. 18151. ISSN: 2045-2322. DOI: 10.1038/s41598-021-97592-x

[22] Michael F. Shlesinger. "Fractal Time and 1/f Noise in Complex Systems". In: *Annals of the New York Academy of Sciences* 504 (1 July 1987), pp. 214–228. ISSN: 1749-6632. DOI: 10.1111/J.1749-6632.1987.TB48734.X

[23] Chao Tang and Per Bak. "Mean field theory of self-organized critical phenomena". In: *Journal of Statistical Physics* 51 (5-6 June 1988), pp. 797–802. ISSN: 00224715. DOI: 10.1007/BF01014884/METRICS

[24] Gerhard Werner. "Fractals in the nervous system: conceptual implications for theoretical neuroscience". In: *Frontiers in Physiology* 1 (2010), p. 15. DOI: 10.3389/fphys.2010.00015

[25] A. Van der Ziel. "Flicker noise in electronic devices". In: *Elsevier* 49 (1979), pp. 225–297. DOI: 10.1016/S0065-2539(08)60768-4

Appendix A FYP Contribution Checklist

	Myself	Others	Comments
Instrument construction			
Basic CA Algorithm	040%	060%	CA is well established in the literature.
Generalised CA Algorithm	100%	000%	I generalised many features of CA to work for countless systems.
Interoperable library	100%	000%	Complimentary analysis and visualisation features.
Sample design and prep.			
BTW model	010%	090%	Minimal adjustments were made/investigated.
OFC model	010%	090%	Minimal adjustments were made/investigated.
Control Strategies	090%	010%	All were invented after some discussion with my supervisor.
Measurements			
BTW model	070%	030%	The new results were compared to existing measurements.
OFC model	070%	030%	The new results were compared to existing measurements.
Control Strategies	100%	000%	Entirely new research.
Analysis			
Figures 2.1, 2.2, 2.3, 4.1, 4.3	000%	100%	Either reproduced from Bak or Olami, or sourced through creative commons.
Other figures and curve fitting	100%	000%	Produced independently with the new library.
Equations	010%	090%	From supervisor and literature, with minor adjustments.
Discussion			
All	060%	040%	My supervisor and I have had a discussed the project at length. The new library is a personal quest.

Appendix B List of Figures

Appendix C List of Tables

Appendix D List of Code

Appendix E Code Snippets

Code E.1: Run method of CellularAutomaton class.

```
def run(self):
  self.progress_bar.mk_bar(self.states, prefix=self.seed)
  while self.data['state'] < self.transient_states + self.stable_states:

    #* START of perturbation time-step, ie. one state of the sample

    for scale in ('start', 'natural', 'mid', 'manual', 'end'):
      init_state   : bool = self.data['state'] == 0
      stable_state : bool = self.data['state'] >= self.transient_states
      match scale:
        case 'start'   : self.start(init_state); continue
        case 'natural' : search_pset, search_area = self.natural(init_state)
        case 'mid'     : self.mid(init_state);   continue
        case 'manual'  : search_pset, search_area = self.manual(init_state)
        case 'end'     : self.end(init_state);   continue
      search_fset = set()
      iter_pset = iter(search_pset)
      duration = 0
      while search_area != 0:

        #* START of event time-step, ie. one iteration of a relaxation event

        search_fset.update(self.rule(next(iter_pset), scale))
        search_area -= 1
        if search_area == 0:
          self.control(f'{scale}_event_series')
```

```
        #? After exhausting one list of cells, restart with relaxed cells.
        iter_pset = iter(search_fset)
        search_area = len(search_fset)
        search_fset = set()
        self.pg = self.fg.copy()
        if search_area != 0:
          duration += 1
          if 'event' in self.output and stable_state:
            store_data()
            self.series.__record__(**self.data['event'])

      #* END of event time-step, ie. one iteration of a relaxation event

    #? A steady state has now been reached!
    if 'perturbation' in self.output and stable_state:
        store_data()
        self.series.__record__(**self.data['pert'])

  #* END of perturbation time-step, ie. one state of the sample
```