UNIVERSITY OF CORDOBA
INSTITUTE OF POSTGRADUATE STUDIES

# UNIVERSITY MASTER'S DEGREE

# PHYSICAL TECHNOLOGY: RESEARCH AND APPLICATIONS

# MASTER'S THESIS

# Exploring the relationship between Hawkes processes and self-organized criticality in living systems

Antonio Rivas Blanco

Tutor(s): Jorge Hidalgo Aguilera and Serena di Santo

Line of research: Modelling of complex systems and their interdisciplinary applications

Córdoba, 07/2024

# Autorización de defensa del Trabajo Fin de Máster

Tutor 1: Jorge Hidalgo Aguilera
Tutor 2: Serena di Santo

INFORMAN: Que el presente trabajo titulado *Exploring the relationship between Hawkes processes and self-organized criticality in living systems* que constituye la memoria del Trabajo Fin de Máster ha sido realizado por Antonio Rivas Blanco y AUTORIZA/N su presentación para que pueda ser defendido en la convocatoria 1ª con fecha 23/07/2024.

Firmado en Córdoba a X de julio de 2024

Tutor 1: Jorge Hidalgo Aguilera                                 Tutor 2: Serena di Santo

# Declaración de autoría

Nombre y apellidos: Antonio Rivas Blanco
DNI: 49832223D

DECLARA

1. Que el trabajo que presenta es totalmente original y que se hace responsable de todo su contenido.

2. Que no ha sido publicado no total ni parcialmente.

3. Que todos los aportes de otros autores/as han sido debidamente referenciados.

4. Que no ha incurrido en fraude científico, plagio o vicios de autoría.

5. Que, en caso de no cumplir los requerimientos anteriores, aceptará las medidas disciplinarias sancionadoras que correspondan.

Firmado en Córdoba a, X de julio de 2024

# Cambiar encabezado y pie de página en .sty

# Contents

# List of Figures

# List of Tables

# Abstract

# Chapter 1

# Introduction

Hawkes processes are a class of point processes that are widely used in the modeling of self-exciting events. They have been applied to a variety of fields, such as seismology, neuroscience, finance, and social networks. The main feature of Hawkes processes is that the intensity or rate of the process is not constant, but depends on the history of the process itself.

# Chapter 2

# Objectives

Ordenar los objetivos una vez escrito el trabajo para que coincidan con como se presenta.

The main objectives of this Master's thesis are:

- To understand what Hawkes processes are, where we can find them, how to generate them computationally and relate them with neuroscience.

- To understand the importance of time binning and reproduce the results of the original paper [1] and compare them with the results obtained in this work.

- ¿Criticality?

- To study the behaviour of a self-exciting process with $n = 2$ and compare it with the case $n = 1$.

- To study the behaviour of an inhibitory and excitatory neuron coupled.

# Chapter 3

# Methodology

# Chapter 4

# Results

This section provides the main results of the investigation. First, the results reproduced from the original paper [1] are presented. Then, the results of the analysis with $n = 2$ are shown. Finally, we have studied the behaviour of an inhibitory and excitatory neuron coupled.

## 4.1 Results from the original paper

The first result is the percolation phase diagram is shown in Figure 4.1. It displays the percolation strength $P_\infty$ versus the resolution parameter $\Delta$.
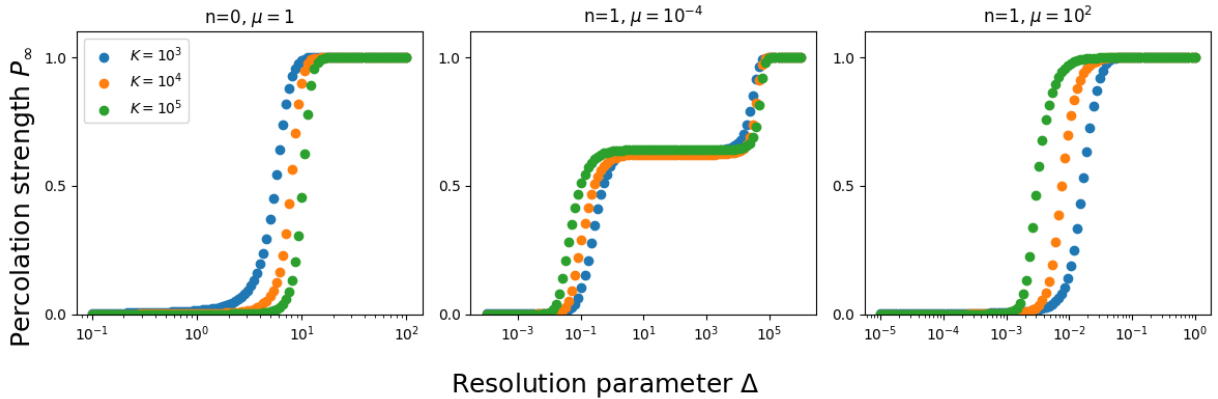


**Figure 4.1.** Percolation phase diagrams for different event number $K$ taking average values of $R = 1000$ realizations.

The first plot configuration is a Markovian ($n = 0$) Poisson process with rate $\mu$. This is the simplest case, where the inter-event time $x = t_i - t_{i-1}$ follows an exponential distribution $P(x_i) = \mu e^{\mu x_i}$. The other two plots are non-Markovian Hawkes processes for $\mu \ll 1$ and $\mu \gg 1$. In one hand, a double transition is observed when $\mu = 10^{-4}$, in the other hand, a single transition occurs when $\mu = 10^2$.

Once we have the phase diagram, we can study avalanche statistics. Given a resolution parameter $\Delta$, we can spot clusters or avalanches of activity. A cluster starts when a neuron fires and ends if the neuron does not fire for a time greater than $\Delta$. We define the size of a cluster as the number of spikes it contains and the duration as the time between the first and last spike. We have studied the avalanches for $K = 10^5$ events and $R = 1000$ realizations to obtain the average values since the process is highly not stationary. We will study the size and duration of the avalanches for the three different

regions of the phase diagram for $\mu = 10^{-4}$ and the two regions of the phase diagram for $\mu = 10^2$. These regions are separated by two thresholds, a pseudocritical threshold $\Delta_1^*$ and the threshold of the second transition at $\Delta_2^*$. We can compute these with the following formulas [1]:

$$\Delta_1^* \simeq \frac{log(K)}{\langle \lambda \rangle} = \frac{log(K)}{\mu + \sqrt{2\mu K}} \tag{4.1}$$

$$\Delta_2^* = \frac{log(K)}{\mu} \tag{4.2}$$

Once we have the thresholds, we can study the avalanches for the different regions of the diagram. The results are shown in Figure 4.2.



For $\mu = 10^{-4}$, the results show a power-law distribution for the size and duration of the avalanches . In the case of duration, the exponent is $\tau = 2$ and for the size, we can notice a transition of the exponent from $\alpha = 2$ to $\alpha = 3/2$ as we increase the resolution parameter $\Delta$. The first exponent corresponds to the universality class of 1D percolation, whereas the second is compatible with the universality class of mean-field branching process. However, if $\Delta \ll \Delta_1^*$, the behaviour is subcritical for the size and duration of the avalanches.

For $\mu = 10^2$, the result shows another power-law distribution for both size and duration of the avalanches unless $\Delta \ll \Delta_2^*$, where the behaviour is subcritical. . In this case, the exponents are $\alpha = \tau = 2$ corresponding to the uni-
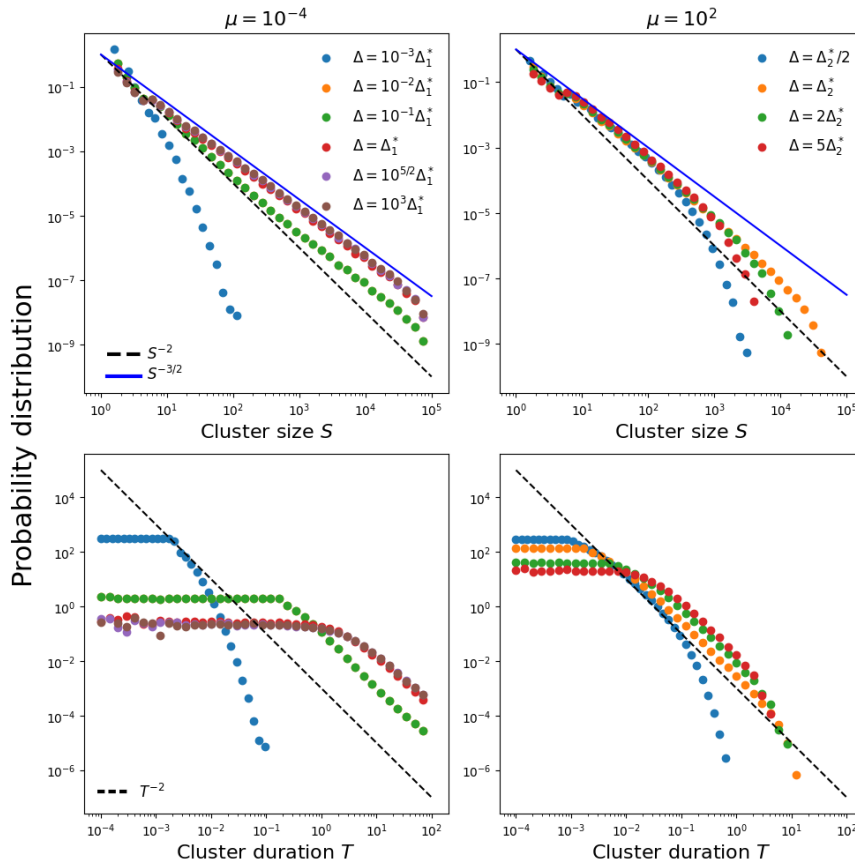
**Figure 4.2.** Avalanche statistics for a self-exciting Hawkes process with $n = 1$ for $K = 10^5$ events averaged over $R = 1000$ realizations.

versality class of 1D percolation.

## 4.2   Results for n=2

In the article, the authors have studied a process which is critical itself because the parameter $n$ is fixed to $n = 1$. We have studied the case $n = 2$ to see if the process is still critical. In the Figure 4.3

6

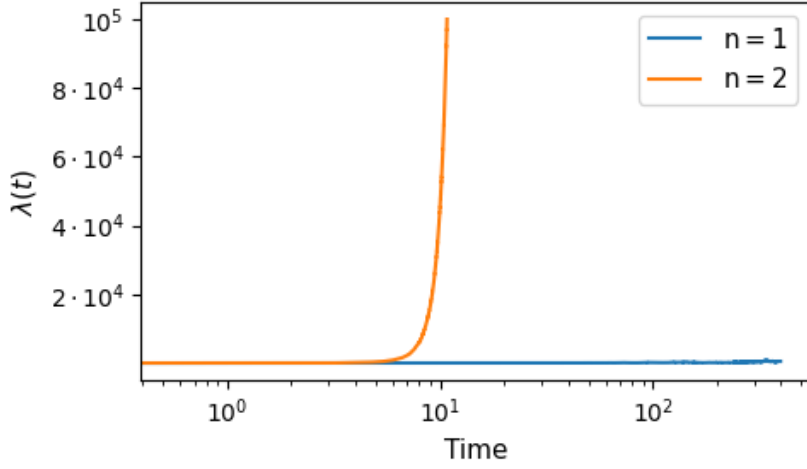two time series for $n = 1$ and $n = 2$ are shown.



**Figure 4.3.** Time series for $n = 1$ and $n = 2$.

Similarly to the previous section, first we obtain the phase diagram in order to observe the transitions. In this case, Eqs 4.1-4.2 are not valid. Therefore, we will obtain this parameter graphically from the phase diagrams shown in Figure 4.4.
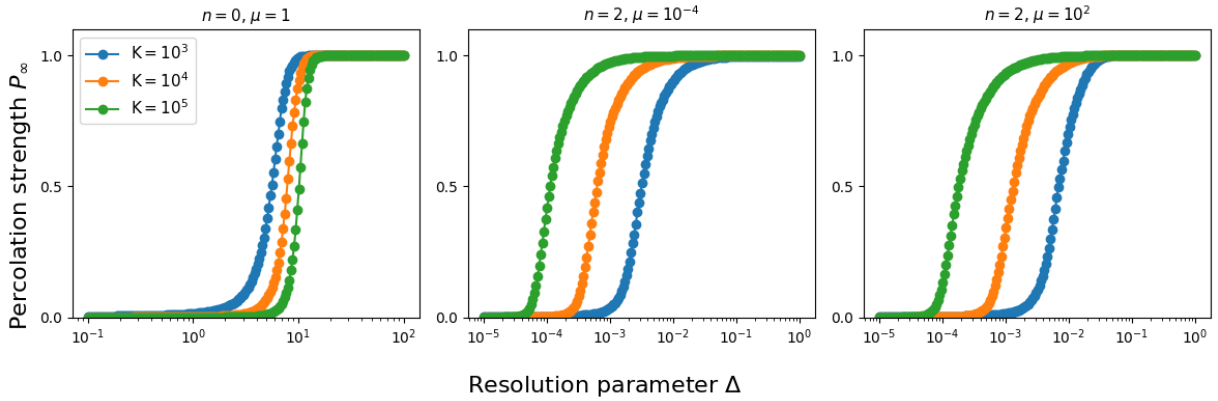


**Figure 4.4.** Percolation phase diagrams for a Hawkes process with $n = 2$.

As we can see, now we have a single transition for $\mu = 10^{-4}$ and $\mu = 10^{2}$ corresponding to 1D percolation, consequently, the exponents for the size and duration should be $\alpha = \tau = 2$. In a similar way, we have studied the avalanches for $K = 10^{5}$ events and $R = 1000$ realizations to obtain the average values. The statistics of the avalanches are shown in Figure 4.5.
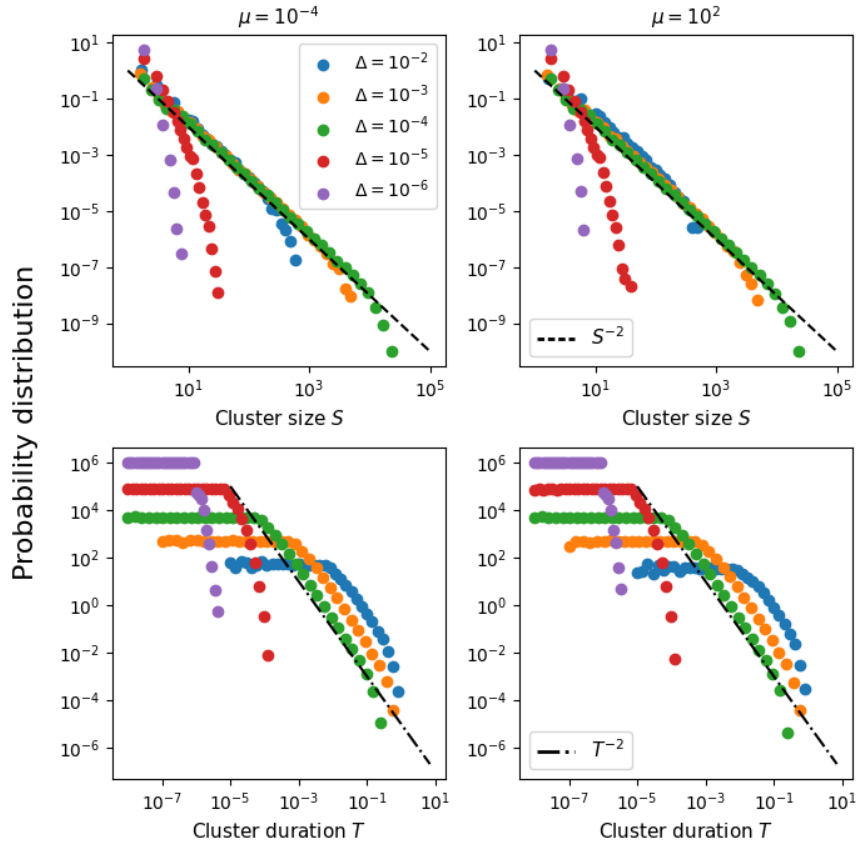
7

**Figure 4.5.** Avalanche statistics for a self-exciting Hawkes process with $n = 2$ for $K = 10^5$ events averaged over $R = 1000$ realizations.

## 4.3 Inhibitory and excitatory neurons coupled

# Chapter 5

# Conclusions

# Bibliography

[1]  Daniele Notarmuzi et al. "Percolation theory of self-exciting temporal processes". In: *Physical Review E* 103.2 (2021), p. L020302.

# Anexo

## REVISAR LOS CÓDIGOS PARA QUE ESTÉN ACTUALIZADOS

**Script 5.1.** *Script* Python con todas las funciones.

```python
import numpy as np
import matplotlib.pyplot as plt

def algorithm(rate, mu, n):
    """
    Algorithm that computes interevent times and Hawkes intensity

    #Output: rate x_k, x_k
    """
    # Paso 1
    u1 = np.random.uniform()
    if mu == 0:
        F1 = np.inf
    else:
        F1 = -np.log(u1) / mu

    # Paso 2
    u2 = np.random.uniform()
    if (rate - mu) == 0:
        G2 = 0
    else:
        G2 = 1 + np.log(u2) / (rate - mu)


    # Paso 3
    if G2 <= 0:
        F2 = np.inf
    else:
        F2 = -np.log(G2)

    # Paso 4
    xk = min(F1, F2)

    # Paso 5
    rate_tk = (rate - mu) * np.exp(-xk) + n + mu
    return rate_tk, xk

def generate_series(K, n, mu):
    """
    Generates temporal series for K Hawkes processes

    ##Inputs:
```

```python
     K: Number of events
     n: Strength of the Hawkes process
     mu: Background intensity

     ##Output:
     times: time series the events
     rate: time series for the intensity
     """
     times_between_events = [0]
     rate = [mu]
     for _ in range(K):
         rate_tk, xk = algorithm(rate[-1], mu, n)
         rate.append(rate_tk)
         times_between_events.append(xk)
     times = np.cumsum(times_between_events)
     return times, rate

def identify_clusters(times, delta):
     """
     Identifies clusters in a temporal series given a resolution parameter delta

     ## Inputs:
     times: temporal series
     delta: resolution parameter

     ## Output:
     clusters: list of clusters
     """
     clusters = []
     current_cluster = []
     for i in range(len(times) - 1):
         if times[i + 1] - times[i] <= delta:
             if not current_cluster:
                 current_cluster.append(times[i])
             current_cluster.append(times[i + 1])
         else:
             if current_cluster:
                 clusters.append(current_cluster)
                 current_cluster = []
     return clusters

def generate_series_perc(K, n, mu):
     """
     Generates temporal series for K Hawkes processes

     ##Inputs:
     K: Number of events
     n: Strength of the Hawkes process
     mu: Background intensity

     ##Output:
     times_between_events: time series the interevent times
     times: time series the events
     rate: time series for the intensity
     """
     times_between_events = [0]
     rate = [mu]
     for _ in range(K):
         rate_tk, xk = algorithm(rate[-1], mu, n)
         rate.append(rate_tk)
         times_between_events.append(xk)
     times = np.cumsum(times_between_events)
```

```python
105        return times_between_events, times, rate
106
107  def calculate_percolation_strength(times_between_events, deltas):
108        percolation_strengths = []
109
110        for delta in deltas:
111            cluster_sizes = []
112            # Initialize the size of the current cluster
113            current_cluster_size = 1 # The first event is always a cluster
114
115            for i in range(len(times_between_events)):
116                if times_between_events[i] <= delta:
117                    current_cluster_size += 1
118                else:
119                    if current_cluster_size > 1: # Only consider clusters with more than one
     event
120                        cluster_sizes.append(current_cluster_size)
121                    # Reset the size of the current cluster
122                    current_cluster_size = 1 # The next event is always a cluster
123
124            # Add the size of the last cluster
125            if current_cluster_size > 1: # Only consider clusters with more than one event
126                cluster_sizes.append(current_cluster_size)
127
128            max_cluster_size = max(cluster_sizes)
129
130            percolation_strengths.append(max_cluster_size / len(times_between_events))
131        return percolation_strengths
132
133  def model(n_max, mu_E, mu_I, tau, n_EE, n_IE, n_EI, n_II, dt):
134        """
135        Solve the equations of the mena field model for a given number of iterations n_max
136
137        Inputs:
138        n_max: number of iterations
139        mu_E: Poisson rate of excitatory neurons
140        mu_I: Poisson rate of inhibitory neurons
141        tau: characteristic time of the system
142        n_EE: influence of excitatory neurons on excitatory neurons
143        n_IE: influence of excitatory neurons on inhibitory neurons
144        n_EI: influence of inhibitory neurons on excitatory neurons
145        n_II: influence of inhibitory neurons on inhibitory neurons
146        dt: time step
147
148        Outputs:
149        time: time series
150        t_events_E: times of events of excitatory neurons
151        t_events_I: times of events of inhibitory neurons
152        rates_E: rates of excitatory neurons
153        rates_I: rates of inhibitory neurons
154        """
155        n_E = n_I = n = 0
156        t_events_E = [0]
157        t_events_I = [0]
158        rates_E = [mu_E]
159        rates_I = [mu_I]
160        time = [0]
161        while n <= n_max:
162            # Excitation neurons
163            l_Enew = rates_E[-1]  + dt * (mu_E- rates_E[-1])/tau
164            if np.random.uniform() < rates_E[-1]*dt:
165                l_Enew += n_EE
```

```
166              t_events_E.append(time[-1]+dt*np.random.uniform())
167              n_E += 1
168          if np.random.uniform() < rates_I[-1]*dt:
169              l_Enew -= n_IE
170              t_events_E.append(time[-1]+dt*np.random.uniform())
171              n_E += 1
172
173          # Inhibition neurons
174          l_Inew = rates_I[-1] + dt * (mu_I- rates_I[-1])/tau
175          if np.random.uniform() < rates_E[-1]*dt:
176              l_Inew += n_EI
177              t_events_I.append(time[-1]+dt*np.random.uniform())
178              n_I += 1
179          if np.random.uniform() < rates_I[-1]*dt:
180              l_Inew -= n_II
181              t_events_I.append(time[-1]+dt*np.random.uniform())
182              n_I += 1
183          rates_E.append(l_Enew)
184          rates_I.append(l_Inew)
185          time.append(time[-1]+dt)
186
187          n = n_E + n_I
188      return time, t_events_E, t_events_I, rates_E, rates_I
```