



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

*“Va por los días en los que quiero un abrazo
y acabo encerrado solo en en cuarto”.
—Piezas*

Cambiar encabezado y pie de página en .sty

Contents

Contents	3
List of Figures	4
List of Tables	5
Abstract. Keywords	6
1 Introduction	7
2 Objectives	8
3 Methodology	9
4 Results	10
4.1 Results from the original paper	10
5 Conclusions	11

List of Figures

4.1 Percolation phase diagrams for different event number K taking average values of $R =$
1000 realizations. 10

List of Tables

Abstract

Chapter 1

Introduction

Chapter 2

Objectives

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_∞ versus the resolution parameter Δ .

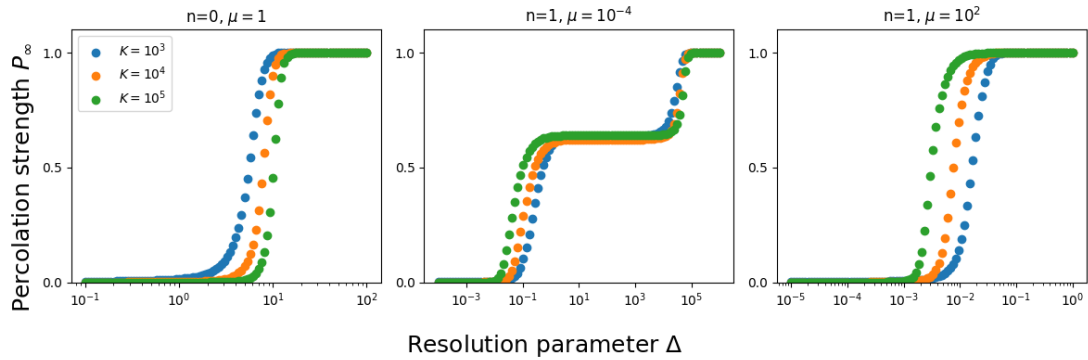


Figure 4.1. Percolation phase diagrams for different event number K taking average values of $R = 1000$ realizations.

The first plot configuration is a Markovian ($n = 0$) Poisson process with rate μ . 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 $\mu = 10^2$.

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.*

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 def algorithm(rate, mu, n):
5     """
6     Algorithm that computes interevent times and Hawkes intensity
7
8     #Output: rate x_k, x_k
9     """
10    # Paso 1
11    u1 = np.random.uniform()
12    if mu == 0:
13        F1 = np.inf
14    else:
15        F1 = -np.log(u1) / mu
16
17    # Paso 2
18    u2 = np.random.uniform()
19    if (rate - mu) == 0:
20        G2 = 0
21    else:
22        G2 = 1 + np.log(u2) / (rate - mu)
23
24    # Paso 3
25    if G2 <= 0:
26        F2 = np.inf
27    else:
28        F2 = -np.log(G2)
29
30    # Paso 4
31    xk = min(F1, F2)
32
33    # Paso 5
34    rateTk = (rate - mu) * np.exp(-xk) + n + mu
35    return rateTk, xk
36
37 def generate_series(K, n, mu):
38     """
39     Generates temporal series for K Hawkes processes
40
41     ##Inputs:
```



```

43 K: Number of events
44 n: Strength of the Hawkes process
45 mu: Background intensity
46
47 ##Output:
48 times: time series the events
49 rate: time series for the intensity
50 """
51 times_between_events = [0]
52 rate = [mu]
53 for _ in range(K):
54     rateTk, xk = algorithm(rate[-1], mu, n)
55     rate.append(rateTk)
56     times_between_events.append(xk)
57 times = np.cumsum(times_between_events)
58 return times, rate
59
60 def identify_clusters(times, delta):
61     """
62     Identifies clusters in a temporal series given a resolution parameter delta
63
64     ## Inputs:
65     times: temporal series
66     delta: resolution parameter
67
68     ## Output:
69     clusters: list of clusters
70     """
71     clusters = []
72     current_cluster = []
73     for i in range(len(times) - 1):
74         if times[i + 1] - times[i] <= delta:
75             if not current_cluster:
76                 current_cluster.append(times[i])
77                 current_cluster.append(times[i + 1])
78             else:
79                 if current_cluster:
80                     clusters.append(current_cluster)
81                     current_cluster = []
82     return clusters
83
84 def generate_series_perc(K, n, mu):
85     """
86     Generates temporal series for K Hawkes processes
87
88     ##Inputs:
89     K: Number of events
90     n: Strength of the Hawkes process
91     mu: Background intensity
92
93     ##Output:
94     times_between_events: time series the interevent times
95     times: time series the events
96     rate: time series for the intensity
97     """
98     times_between_events = [0]
99     rate = [mu]
100     for _ in range(K):
101         rateTk, xk = algorithm(rate[-1], mu, n)
102         rate.append(rateTk)
103         times_between_events.append(xk)
104     times = np.cumsum(times_between_events)

```

```

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

```