



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 1^a 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

Contents	IV
List of Figures	V
List of Tables	VI
Abstract. Keywords	1
1 Introduction	2
2 Objectives	4
3 Methodology	5
4 Results	6
4.1 Results from the original paper	6
4.2 Results for $n=2$	8
4.3 Inhibitory and excitatory neurons coupled	10
5 Conclusions	11

List of Figures

1.1	Representation of a point process. The intensity function $\lambda(t)$ is a time-dependent function.	2
4.1	Percolation phase diagrams for different event number K taking average values of $R = 1000$ realizations.	6
4.2	Avalanche statistics for a self-exciting Hawkes process with $n = 1$ for $K = 10^5$ events averaged over $R = 1000$ realizations.	7
4.3	Time series for $n = 1$ and $n = 2$	8
4.4	Percolation phase diagrams for a Hawkes process with $n = 2$	8
4.5	Avalanche statistics for a self-exciting Hawkes process with $n = 2$ for $K = 10^5$ events averaged over $R = 1000$ realizations.	9

List of Tables

Abstract

Chapter 1

Introduction

Within the large framework of complex systems, stochastic processes lend us a hand to decypher properties of living systems, bridging randomness with structured behaviour. These processes are used to model the dynamics of systems which evolve randomly in time. This is why they are ideal for describing natural phenomena such as the spread of diseases [1], social networks [2] or ecological systems [3]. Mathematically, a stochastic process is a collection of random variables [4], generally ordered in time $\{X_t\}_{t \in T}$, where t is the time and X_t is the system state at time t . T is the time index set, which can be discrete or continuous, in this work we will focus on the discrete case because we are interested in the study of point (Hawkes) processes for modeling neurons.

Point processes are a type of stochastic process that describe the occurrence of events in time or space. We will be interested in time point processes because we are going to model the spiking activity of neurons. For our purposes, they will be characterized by two parameters, the time of occurrence of the events t_k and the intensity or rate of occurrence of these events λ . This rate tells us how likely it is that an event occurs at time t given the history of the process (probability density function, PDF) as pictured in Figure 1.1.

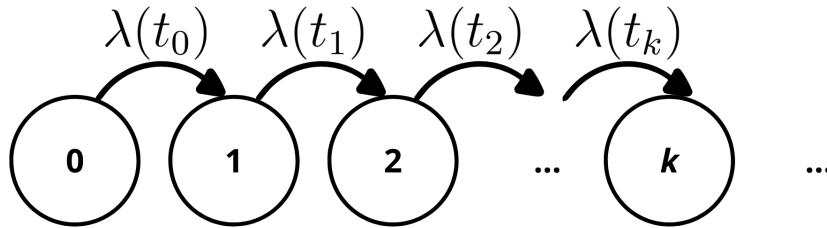


Figure 1.1. Representation of a point process. The intensity function $\lambda(t)$ is a time-dependent function.

In general, the rate is a function of the history of the process, which makes the process non-Markovian. But in our case, it will be a Markovian process, which means that the rate depends only on the last event that occurred as we will see. An example of a Markovian point process is the Poisson process, which is a simple and one of the most studied point processes because they are present in many everyday situations such as the arrival of customers at a store, occurrence of defects on a Production line. They are also present in physics, for instance, the decay of radioactive particles or the arrival of photons at a detector. These processes are characterized by a constant rate of occurrence of events

making them memoryless and Markovian. [MIRAR INFO DE POISSON EN WIKIPEDIA](#)

In most cases, the motivation of study of point processes is counting them, but in our case we also are interested in the time of occurrence of the events which will let us define bursts or avalanches of activity that we will use to describe the dynamics of the system.

PONER AQUÍ ALGUNA FIGURA GENERADA POR MI

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 [5] 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 [5] 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 .

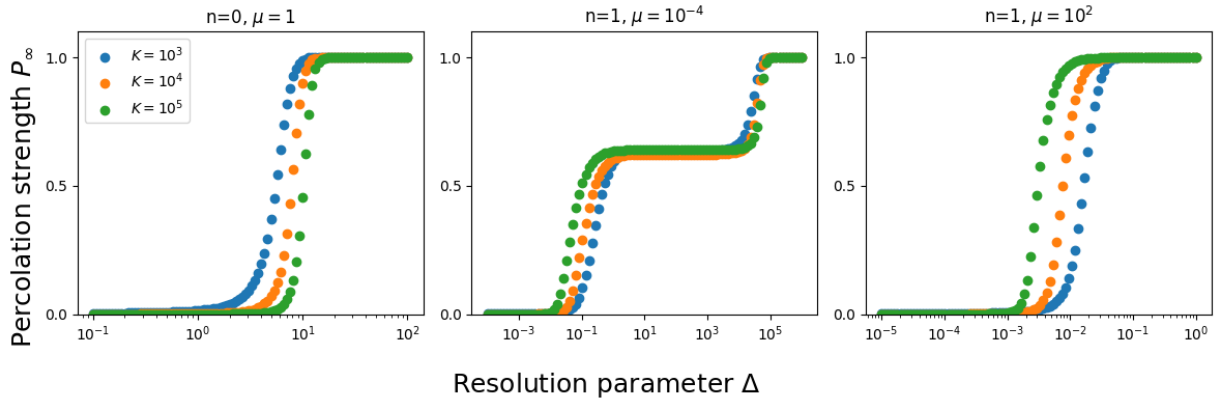


Figure 4.1. Percolation phase diagrams for different event number K taking average values of $R = 1000$ realizations.

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 Hawkes processes for $\mu \ll 1$ and $\mu \gg 1$ that are also Markovian as we have chosen an exponential kernel (REFERENCIAR AQUÍ A LA PARTE EN LA QUE SE EXPLICA EN METODOLOGÍA.) . 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 Δ , 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 Δ . 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 Δ_1^* and the threshold of the second transition at Δ_2^* . We can compute these with the following formulas [5]:

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

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

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.

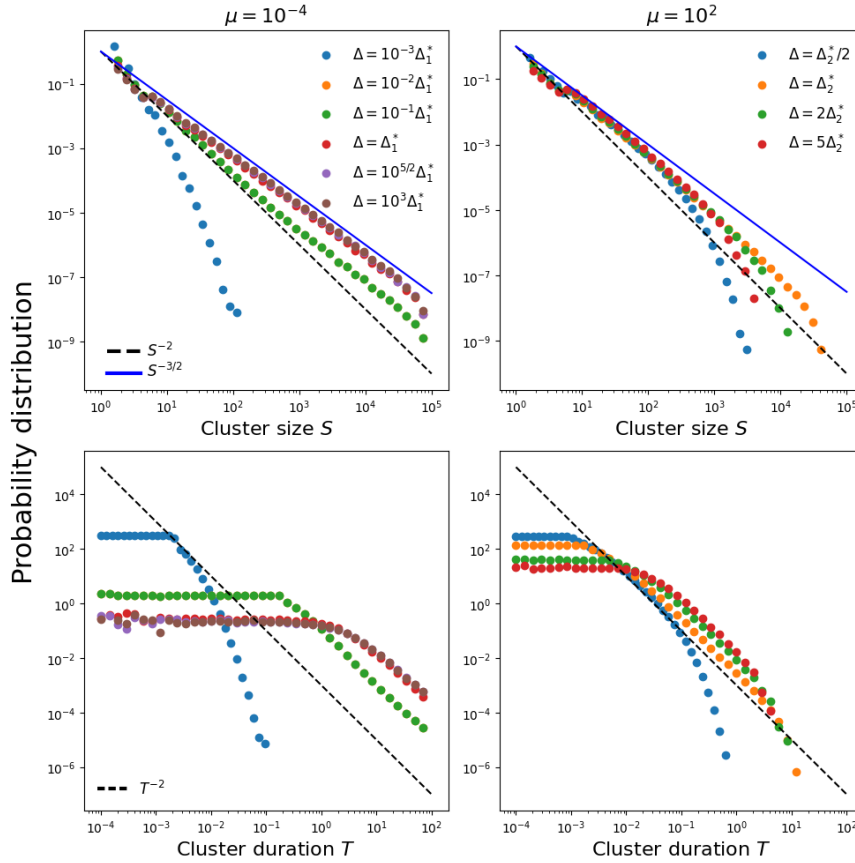


Figure 4.2. Avalanche statistics for a self-exciting Hawkes process with $n = 1$ for $K = 10^5$ events averaged over $R = 1000$ realizations.

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 Δ . 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-

versality class of 1D percolation.

HABLAR AQUÍ DE LA INFLUENCIA DEL MENOR NÚMERO DE EVENTOS, SE SIGUE PRODUCIENDO LA TRANSICIÓN, PERO PARA OTROS VALORES DE DELTA

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

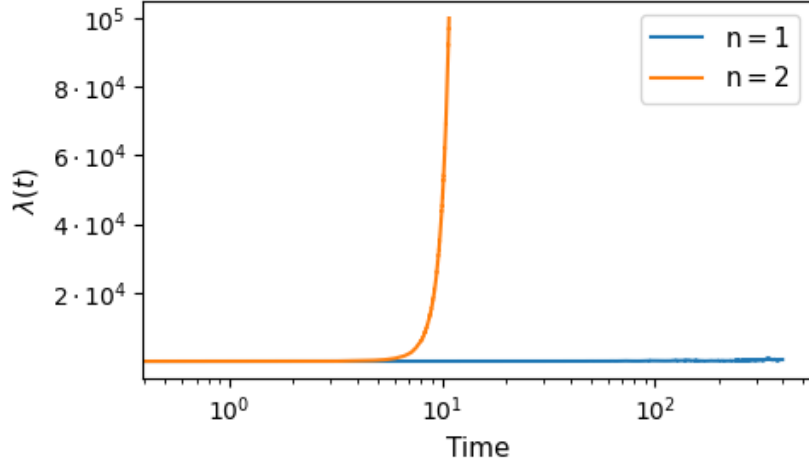


Figure 4.3. Time series for $n = 1$ and $n = 2$.

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.

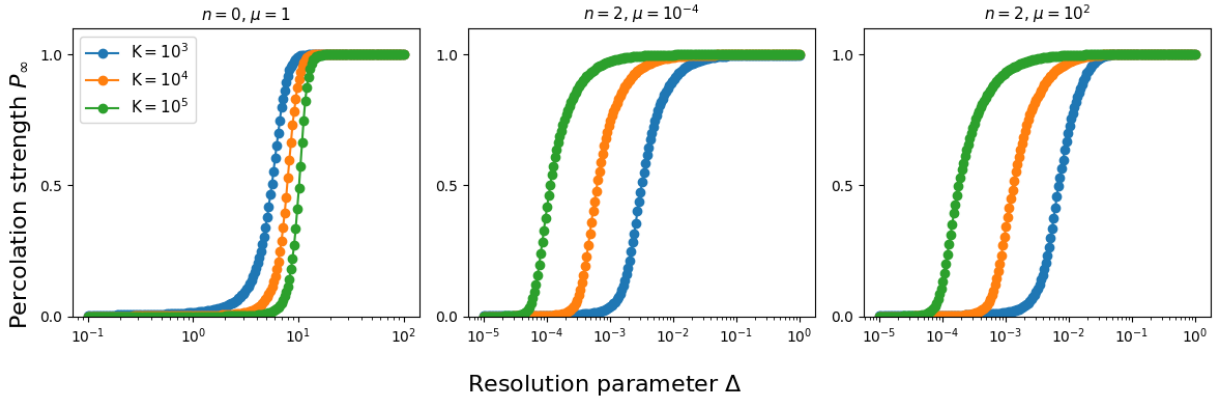


Figure 4.4. Percolation phase diagrams for a Hawkes process with $n = 2$.

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.

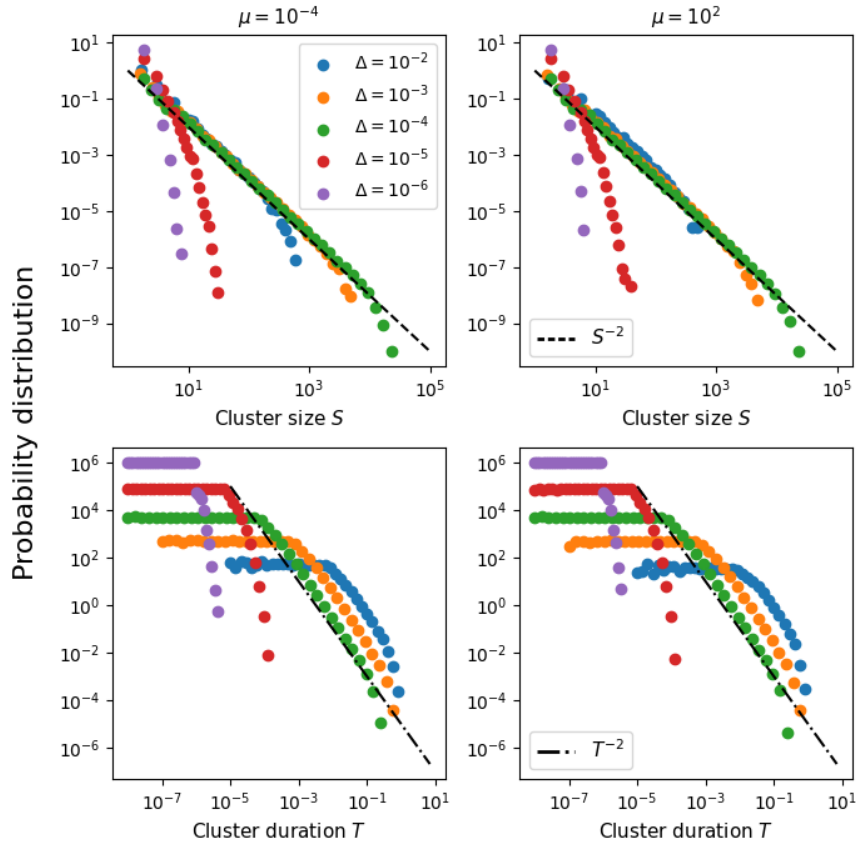


Figure 4.5. Avalanche statistics for a self-exciting Hawkes process with $n = 2$ for $K = 10^5$ events averaged over $R = 1000$ realizations.

4.3 Inhibitory and excitatory neurons coupled

Chapter 5

Conclusions

Bibliography

- [1] Gerardo Chowell et al. “Mathematical models to characterize early epidemic growth: A review”. In: *Physics of life reviews* 18 (2016), pp. 66–97.
- [2] Claudio Castellano, Santo Fortunato, and Vittorio Loreto. “Statistical physics of social dynamics”. In: *Reviews of modern physics* 81.2 (2009), p. 591.
- [3] Sandro Azaele et al. “Statistical mechanics of ecological systems: Neutral theory and beyond”. In: *Reviews of Modern Physics* 88.3 (2016), p. 035003.
- [4] Alan J McKane. *Stochastic Processes*. 2009.
- [5] 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 for a self-exciting
7     process
8
9     #Output: rate x_k, x_k
10    """
11    # 1st step
12    u1 = np.random.uniform()
13    if mu == 0:
14        F1 = np.inf
15    else:
16        F1 = -np.log(u1) / mu
17
18    # 2nd step
19    u2 = np.random.uniform()
20    if (rate - mu) == 0:
21        G2 = 0
22    else:
23        G2 = 1 + np.log(u2) / (rate - mu)
24
25    # 3rd step
26    if G2 <= 0:
27        F2 = np.inf
28    else:
29        F2 = -np.log(G2)
30
31    # 4th step
32    xk = min(F1, F2)
33
34    # 5th step
35    rateTk = (rate - mu) * np.exp(-xk) + n + mu
36    return rateTk, xk
37
38 def generate_series(K, n, mu):
39     """
40     Generates temporal series for K Hawkes processes
41
```

```

42  ##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     """
109     Calculate the percolation strength for a given set of deltas (resolution parameters)
110
111     ## Inputs:
112     times_between_events: time series of interevent times
113     deltas: list of resolution parameters
114
115     ## Output:
116     percolation_strengths: list of percolation strengths
117     """
118
119     percolation_strengths = []
120
121     for delta in deltas:
122         cluster_sizes = []
123         current_cluster_size = 1 # The first event is always a cluster
124
125         for time in times_between_events:
126             if time < delta:
127                 current_cluster_size += 1
128             else:
129                 if current_cluster_size > 1: # Only consider clusters with more than one
event
130                     cluster_sizes.append(current_cluster_size)
131                     current_cluster_size = 1 # The next event is always a cluster
132
133                 if current_cluster_size > 1: # Consider the last cluster if it ends at the last
event
134                     cluster_sizes.append(current_cluster_size)
135
136                 if len(cluster_sizes) != 0: # Check if cluster_sizes is not empty to avoid
errors
137                     max_cluster_size = max(cluster_sizes)
138                 else:
139                     max_cluster_size = 0
140
141                 percolation_strengths.append(max_cluster_size / len(times_between_events))
142
143     return percolation_strengths
144
145 """
146 def calculate_percolation_strength(times_between_events, deltas):
147     percolation_strengths = []
148
149     for delta in deltas:
150         cluster_sizes = []
151         # Initialize the size of the current cluster
152         current_cluster_size = 1 # The first event is always a cluster
153
154         for i in range(len(times_between_events)):
155             if times_between_events[i] <= delta:
156                 current_cluster_size += 1
157             else:
158                 if current_cluster_size > 1: # Only consider clusters with more than one
event
159                     cluster_sizes.append(current_cluster_size)
160                     # Reset the size of the current cluster
161                     current_cluster_size = 1 # The next event is always a cluster

```

```

162     # Add the size of the last cluster
163     if current_cluster_size > 1: # Only consider clusters with more than one event
164         cluster_sizes.append(current_cluster_size)
165
166     max_cluster_size = max(cluster_sizes)
167
168     percolation_strengths.append(max_cluster_size / len(times_between_events))
169 return percolation_strengths"""
170
171 def model(n_max, mu_E, mu_I, tau, n_EE, n_IE, n_EI, n_II, dt):
172     """
173     Solve the equations of the mena field model for a given number of iterations n_max
174
175     Inputs:
176     n_max: number of iterations
177     mu_E: Poisson rate of excitatory neurons
178     mu_I: Poisson rate of inhibitory neurons
179     tau: characteristic time of the system
180     n_EE: influence of excitatory neurons on excitatory neurons
181     n_IE: influence of excitatory neurons on inhibitory neurons
182     n_EI: influence of inhibitory neurons on excitatory neurons
183     n_II: influence of inhibitory neurons on inhibitory neurons
184     dt: time step
185
186     Outputs:
187     time: time series
188     t_events_E: times of events of excitatory neurons
189     t_events_I: times of events of inhibitory neurons
190     rates_E: rates of excitatory neurons
191     rates_I: rates of inhibitory neurons
192     """
193     n_E = n_I = n = 0
194     t_events_E = [0]
195     t_events_I = [0]
196     rates_E = [mu_E]
197     rates_I = [mu_I]
198     time = [0]
199     while n <= n_max:
200         # Excitation neurons
201         l_Enew = rates_E[-1] + dt * (mu_E - rates_E[-1])/tau
202         if np.random.uniform() < rates_E[-1]*dt:
203             l_Enew += n_EE
204             t_events_E.append(time[-1]+dt*np.random.uniform())
205             n_E += 1
206         if np.random.uniform() < rates_I[-1]*dt:
207             l_Enew -= n_IE
208             t_events_E.append(time[-1]+dt*np.random.uniform())
209             n_E += 1
210
211         # Inhibition neurons
212         l_Inew = rates_I[-1] + dt * (mu_I - rates_I[-1])/tau
213         if np.random.uniform() < rates_E[-1]*dt:
214             l_Inew += n_EI
215             t_events_I.append(time[-1]+dt*np.random.uniform())
216             n_I += 1
217         if np.random.uniform() < rates_I[-1]*dt:
218             l_Inew -= n_II
219             t_events_I.append(time[-1]+dt*np.random.uniform())
220             n_I += 1
221         rates_E.append(l_Enew)
222         rates_I.append(l_Inew)
223

```

```

224         time.append(time[-1]+dt)
225
226         n = n_E + n_I
227     return time, t_events_E, t_events_I, rates_E, rates_I
228
229 def identify_clusters_model(times, delta):
230     """
231     Identifies clusters in a temporal series given a resolution parameter delta
232     Computes the size and duration of clusters
233
234     ## Inputs:
235     times: temporal series
236     delta: resolution parameter
237
238     ## Output:
239     clusters: list of clusters
240     clusters_sizes: list of sizes of clusters
241     clusters_times: list of durations of clusters
242     """
243     clusters = []
244     current_cluster = []
245     for i in range(len(times) - 1):
246         if times[i + 1] - times[i] <= delta:
247             if not current_cluster:
248                 current_cluster.append(times[i])
249                 current_cluster.append(times[i + 1])
250             else:
251                 if current_cluster:
252                     clusters.append(current_cluster)
253                     current_cluster = []
254
255     clusters_sizes = [len(cluster) for cluster in clusters]
256     clusters_times = [cluster[-1] - cluster[0] for cluster in clusters]
257     return clusters, clusters_sizes, clusters_times
258
259 def bivariate_algorithm(rate1, rate2, muE, muI, nEE, nII, nEI, nIE):
260     """
261     Algorithm that computes interevent times and Hawkes intensity for a bivariate Hawkes
262     process
263
264     #Inputs:
265     rate1: Previous excitation rate
266     rate2: Previous inhibition rate
267     nEE: "Strength" of the autoexcitation process
268     nII: "Strength" of the autoinhibition process
269     nEI: "Strength" of the excitation to the inhibition
270     nIE: "Strength" of the inhibition to the excitation
271     muE: Background intensity of the excitation
272     muI: Background intensity of the inhibition
273
274     #Output: ratex_k, x_k, reaction (0 for excitatory events and 1 for inhibitory events)
275     """
276     _, xk1 = algorithm(rate1, muE, nEE)
277     _, xk2 = algorithm(rate2, muI, nII)
278
279     xks = [xk1, xk2]
280
281     if xk1 < 0:
282         print(xk1)
283         xk1 = 0
284     if xk2 < 0:

```



```

285     print(xk2)
286     xk2 = 0
287
288     reaction = np.argmin(xks)
289
290     if reaction == 0:
291         rate1_tk = (rate1 - muE) * np.exp(-xk1) + nEE + muE
292         rate2_tk = (rate2 - muI) * np.exp(-xk1) + nEI + muI
293     else:
294         rate1_tk = (rate1 - muE) * np.exp(-xk2) + nIE + muE
295         rate2_tk = (rate2 - muI) * np.exp(-xk2) + nII + muI
296
297     if rate1_tk <= muE:
298         rate1_tk = muE
299     if rate2_tk <= muI:
300         rate2_tk = muI
301
302     xk = xks[reaction]
303
304     return rate1_tk, rate2_tk, xk, reaction
305
306 def generate_series_bivariate(K, nEE, nII, nEI, nIE, muE, muI):
307     """
308     Generates temporal series for K bivariate Hawkes processes
309
310     ##Inputs:
311     K: Number of events
312     nEE: "Strength" of the autoexcitation process
313     nII: "Strength" of the autoinhibition process
314     nEI: "Strength" of the excitation to the inhibition
315     nIE: "Strength" of the inhibition to the excitation
316     muE: Background intensity of the excitation
317     muI: Background intensity of the inhibition
318
319     ##Output:
320     times_between_events: time series the interevent times
321     times: time series the events
322     rate1: time series for the intensity of process 1 (Excitation)
323     rate2: time series for the intensity of process 2 (Inhibition)
324     reactions: list the event type (0 for excitation. 1 for inhibition)
325     """
326     times_between_events = [0]
327     rate1 = [muE]
328     rate2 = [muI]
329     reactions = []
330     for _ in range(K):
331         rate1_tk, rate2_tk, xk, reaction = bivariate_algorithm(rate1[-1], rate2[-1], muE,
332         muI, nEE, nII, nEI, nIE)
333         rate1.append(rate1_tk)
334         rate2.append(rate2_tk)
335         reactions.append(reaction)
336         times_between_events.append(xk)
337     times = np.cumsum(times_between_events)
338     return times_between_events, times, rate1, rate2, reactions

```