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

Within the large framework of complex systems, stochastic processes lend us a hand to decypher properties of living systems, bridging randomness with structured behaviour. This 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 tell us how likely is that an event occurs at time t given the history of the process (probability density function, PDF) as pictured in Figure 1.1.

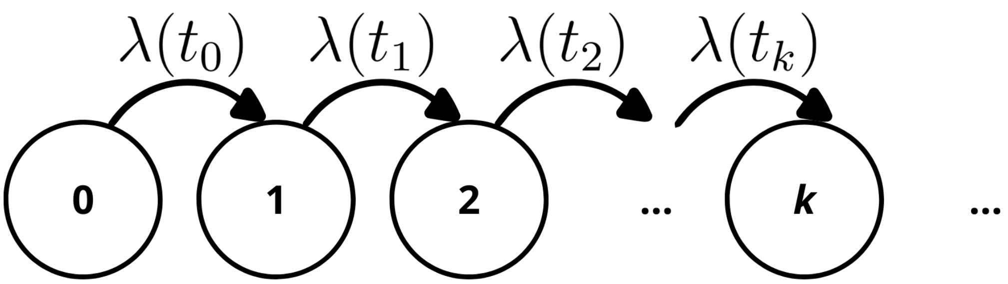

Figure 1.1. Representation of a point process. The intensity function $\lambda(t)$ is a time-dependent function.

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 some physics phenomena, for instance, the decay of radioactive particles or the arrival of photons at a detector. These processes are characterized by a rate of occurrence of events λ. The dynamics of these processes are described by the Poisson distribution which is the probability distribution of a random variable N such that the probability that $N = n$ is:

$$P(N = n) = \frac{\lambda^n}{n!} e^{-\lambda}. \tag{1.1}$$

Furthermore, the mean value and the variance of the distribution are also equal to λ. Poisson processes can be homogeneous or inhomogeneous, depending on whether the rate is constant or time-dependent. In Figure 1.2 we can see an example of a homogeneous Poisson process.

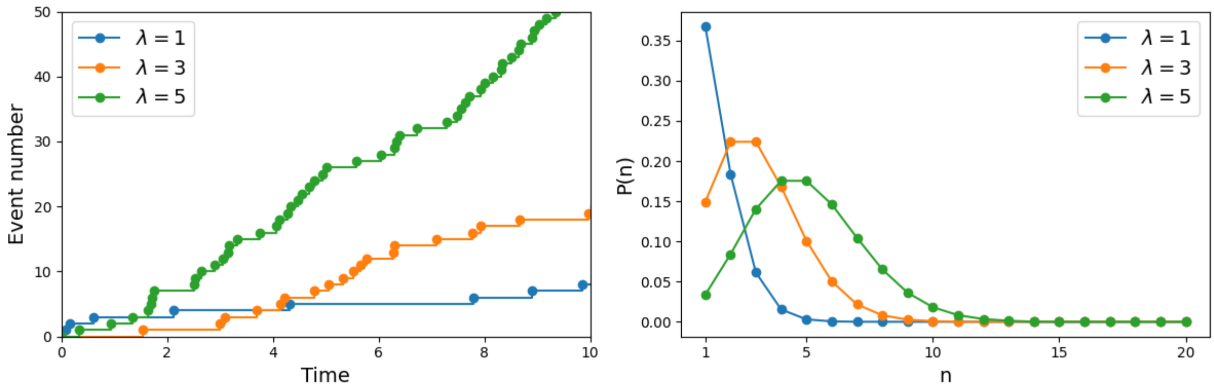

Figure 1.2. Left: event number in time for different rates. Right: Probability of having a certain number of events for different rates.

On the other hand if we consider a non-homogeneous Poisson process, the rate is a function of time, $\lambda(t)$, which is the case of the Hawkes process. The rate can be written in several ways [5, 6, 7, 8]. We will use the the expression from [5]:

$$\lambda(t|t_1, \ldots, t_k) = \mu + n \sum_{i=1}^{k} \phi(t - t_i), \tag{1.2}$$

where μ is the background rate of a homogeneous Poisson process, n is a parameter that controls the strength the self-excitation, and $\phi(t)$ is the kernel function that describes the influence of the past events on the rate of occurrence of the events. The kernel function is a non-negative and monotonically non-increasing function that integrates to 1. Typical choices for the kernel function are the exponential or the power-law functions. In this work we will focus on the exponential kernel. From Eq 1.2 we can see that the rate depends on the history of the process, making it non-Markovian in general, but with an exponential kernel, the process becomes Markovian. The kernel function can be written as: $\phi(t) = \sum_{t_i<t} \alpha e^{-\beta(t-t_i)}$ so the rate becomes:

$$
\begin{aligned}
\lambda(t) &= \mu + \sum_{t_i<t} \alpha e^{-\beta(t-t_i)} \\
&= \mu + \sum_{\underbrace{t_i < t_k}_{t_k\text{: last event}}} \alpha e^{-\beta(t-t_k+t_k-t_i)} \\
&= \mu + e^{-\beta(t-t_k)} \underbrace{\sum_{t_i<t_k} \alpha e^{-\beta(t_k-t_i)}}_{\lambda(t_k)} \\
&= \mu + e^{-\beta(t-t_k)} \left(\lambda(t_k) - \mu + n\right).
\end{aligned}
\tag{1.3}
$$

Where we have used the following expression for the rate of the Hawkes process at time t_k:

$$
\lambda(t_k) = \mu + \sum_{t_i<t_k} \alpha e^{-\beta(t_k-t_i)} \Rightarrow \sum_{t_i<t_k} \alpha e^{-\beta(t_k-t_i)} = \lambda(t_k) - \mu + \alpha \tag{1.4}
$$

Despite being a Markovian process, it is still an inhomogeneous Poisson process because the rate is not constant. In addition, it is a self-exciting process, which means that the occurrence of an event increases the probability of the occurrence of another event. This is why it is used to model the spiking activity of neurons, where the occurrence of a spike increases the probability of the occurrence of another spike. This self-excitation will enable the appearance of bursts of activity that we will measure. The parameters chosen for the kernel function will be $\alpha = \beta = 1$ and we will vary the background rate μ from values much smaller than 1 to values much larger than 1. In Figures 1.3 and 1.4 we can see typical diagrams of Hawkes processes with these parameters.

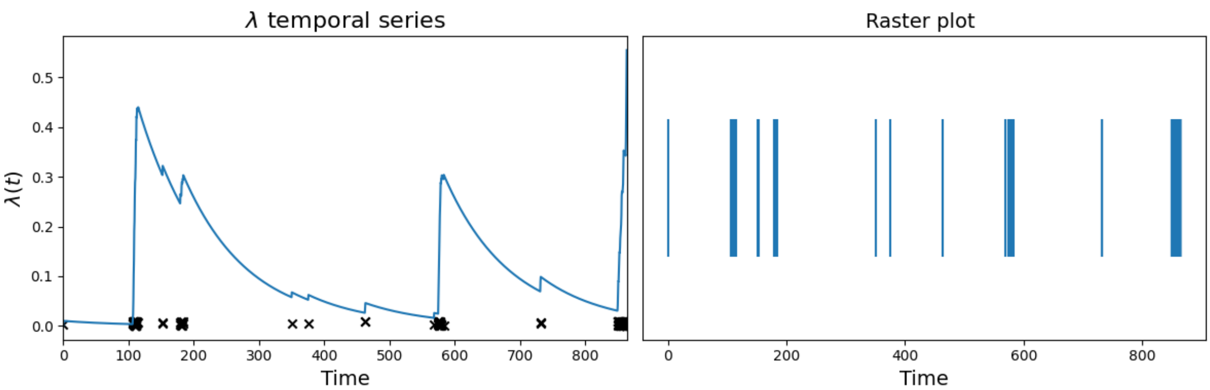

Figure 1.3. On the left, a temporal series of $K = 150$ events of a Hawkes process with $\mu = 0.01$, on the right, a raster plotof the same process.

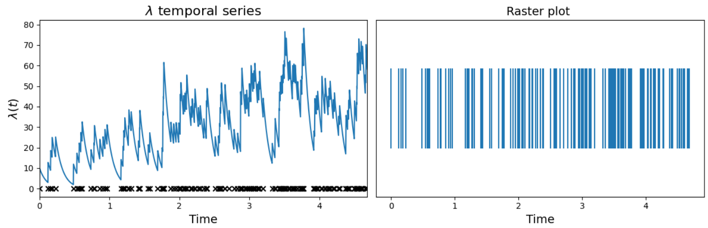

Figure 1.4. On the left, a temporal series of $K = 150$ events of a Hawkes process with $\mu = 0.01$, on the right, a raster plot of the same process.

In most cases, the motivation of study of point processes is counting the events, 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.

Unless otherwise stated, $n = 1$ HABLAR DE CRITICIDAD A PARTIR DE AQUÍ

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.

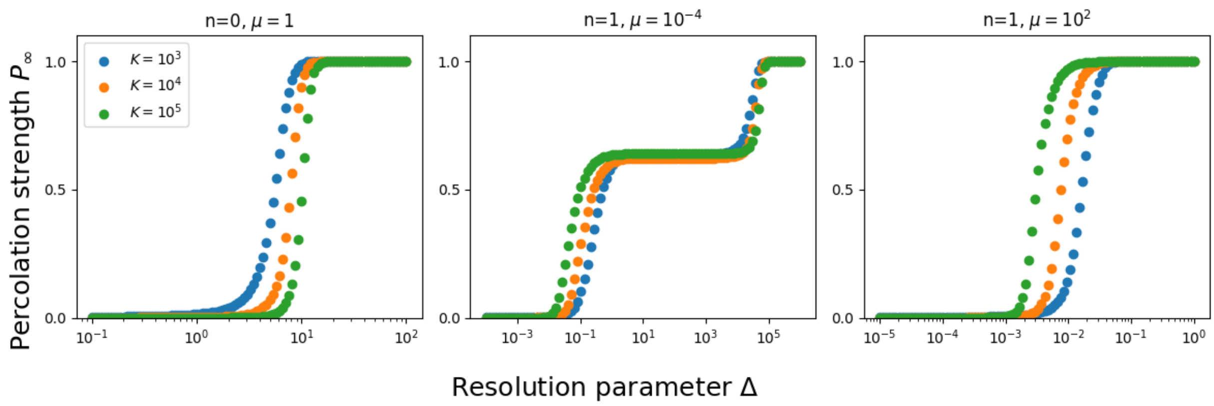

Figure 4.1. Percolation phase diagrams for different event number K taking average values of $R = 1000$ realizations.

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]:

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

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

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.

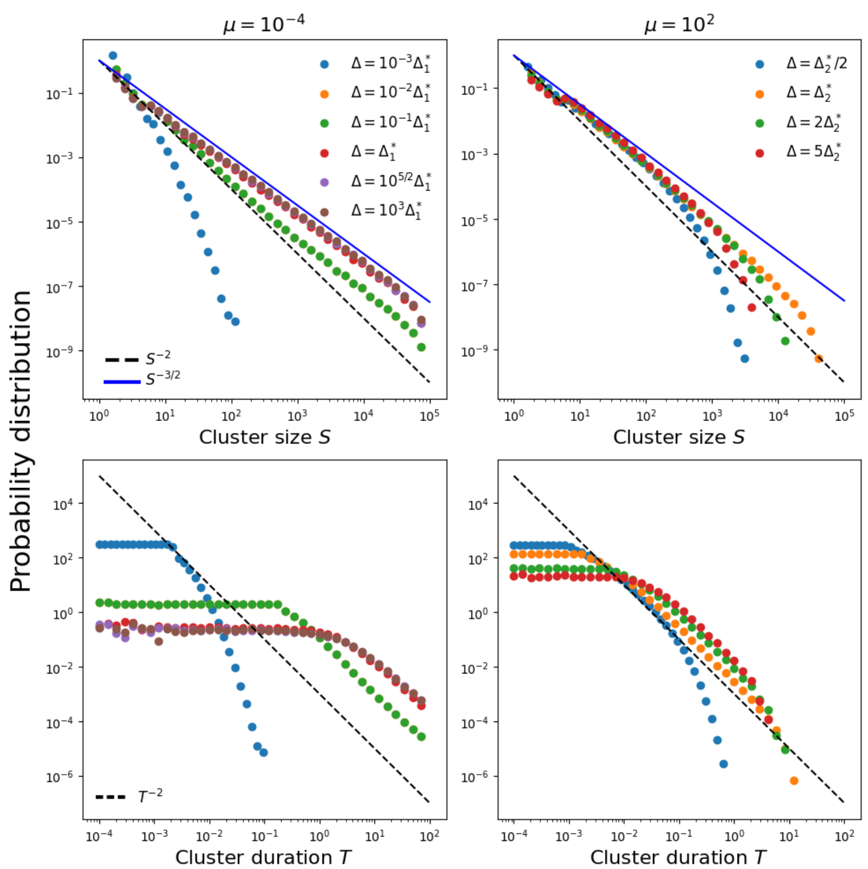

Figure 4.2. Avalanche statistics for a self-exciting Hawkes process with $n = 1$ for $K = 10^5$ events averaged over $R = 1000$ realizations.

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 universality 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.

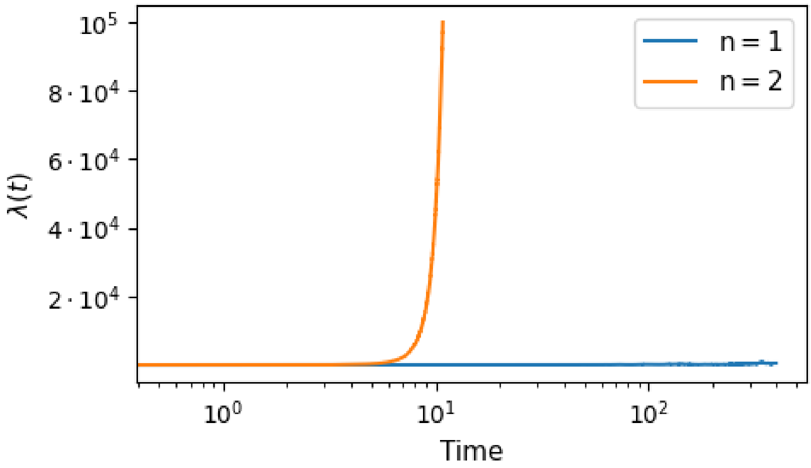

Figure 4.3. Time series for $n = 1$ and $n = 2$.

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.

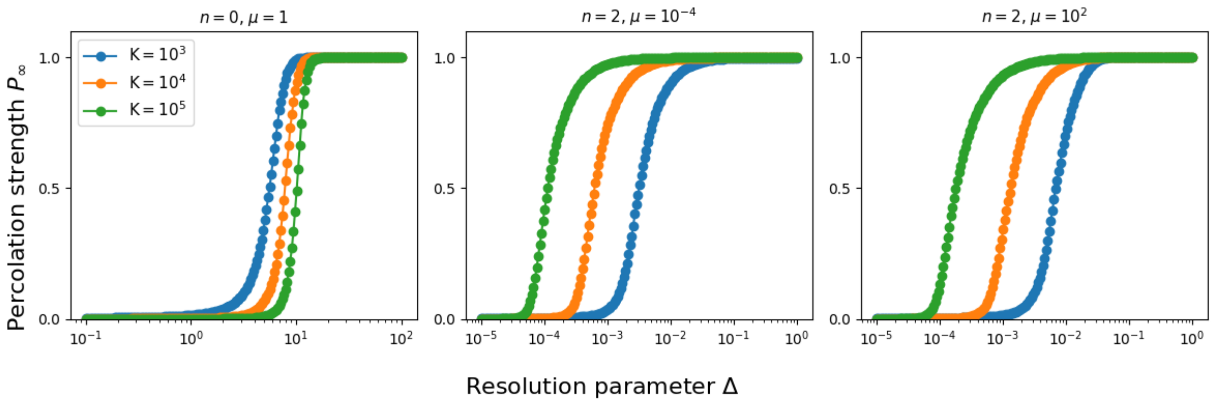

Figure 4.4. Percolation phase diagrams for a Hawkes process with $n = 2$.

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.

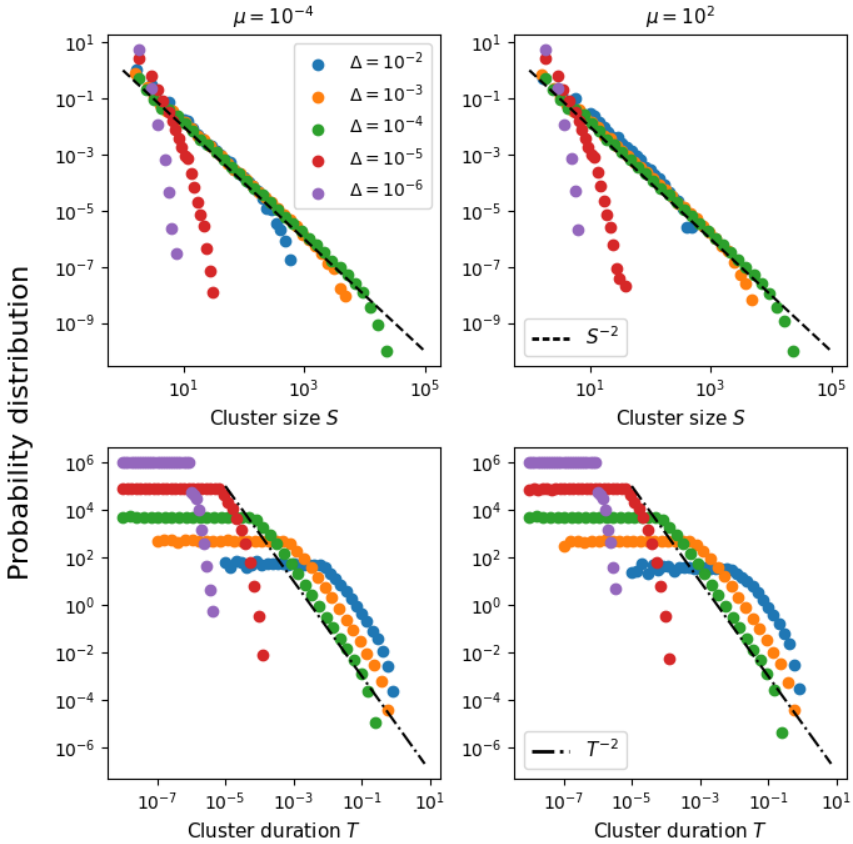

Figure 4.5. Avalanche statistics for a self-exciting Hawkes process with $n = 2$ for $K = 10^5$ events averaged over $R = 1000$ realizations.

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.

[6] Kiyoshi Kanazawa and Didier Sornette. “Ubiquitous power law scaling in nonlinear self-excited Hawkes processes”. In: *Physical review letters* 127.18 (2021), p. 188301.

[7] Angelos Dassios and Hongbiao Zhao. “Exact simulation of Hawkes process with exponentially decaying intensity”. In: (2013).

[8] Patrick J Laub, Young Lee, and Thomas Taimre. *The elements of Hawkes processes.* Springer, 2021.

Anexo

REVISAR LOS CÓDIGOS PARA QUE ESTÉN ACTUALIZADOS

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

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

def algorithm(rate, mu, n):
    """
    Algorithm that computes interevent times and Hawkes intensity for a self-exciting process

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

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


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

    # 4th step
    xk = min(F1, F2)

    # 5th step
    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:
    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)
    return times_between_events, times, rate

def calculate_percolation_strength(times_between_events, deltas):
    """
    Calculate the percolation strength for a given set of deltas (resolution parameters)

    ## Inputs:
    times_between_events: time series of interevent times
    deltas: list of resolution parameters

    ## Output:
    percolation_strengths: list of percolation strengths
    """

    percolation_strengths = []

    for delta in deltas:
        cluster_sizes = []
        current_cluster_size = 1 # The first event is always a cluster

        for time in times_between_events:
            if time < delta:
                current_cluster_size += 1
            else:
                if current_cluster_size > 1: # Only consider clusters with more than one event
                    cluster_sizes.append(current_cluster_size)
                current_cluster_size = 1 # The next event is always a cluster

        if current_cluster_size > 1: # Consider the last cluster if it ends at the last event
            cluster_sizes.append(current_cluster_size)

        if len(cluster_sizes) != 0:  # Check if cluster_sizes is not empty to avoid errors
            max_cluster_size = max(cluster_sizes)
        else:
            max_cluster_size = 0

        percolation_strengths.append(max_cluster_size / len(times_between_events))

    return percolation_strengths


"""def calculate_percolation_strength(times_between_events, deltas):
    percolation_strengths = []

    for delta in deltas:
        cluster_sizes = []
        # Initialize the size of the current cluster
        current_cluster_size = 1 # The first event is always a cluster

        for i in range(len(times_between_events)):
            if times_between_events[i] <= delta:
                current_cluster_size += 1
            else:
                if current_cluster_size > 1: # Only consider clusters with more than one event
                    cluster_sizes.append(current_cluster_size)
                # Reset the size of the current cluster
                current_cluster_size = 1 # The next event is always a cluster
```

```

        # Add the size of the last cluster
        if current_cluster_size > 1: # Only consider clusters with more than one event
            cluster_sizes.append(current_cluster_size)

        max_cluster_size = max(cluster_sizes)

        percolation_strengths.append(max_cluster_size / len(times_between_events))
    return percolation_strengths"""

def model(n_max, mu_E, mu_I, tau, n_EE, n_IE, n_EI, n_II, dt):
    """
    Solve the equations of the mena field model for a given number of iterations n_max

    Inputs:
    n_max: number of iterations
    mu_E: Poisson rate of excitatory neurons
    mu_I: Poisson rate of inhibitory neurons
    tau: characteristic time of the system
    n_EE: influence of excitatory neurons on excitatory neurons
    n_IE: influence of excitatory neurons on inhibitory neurons
    n_EI: influence of inhibitory neurons on excitatory neurons
    n_II: influence of inhibitory neurons on inhibitory neurons
    dt: time step

    Outputs:
    time: time series
    t_events_E: times of events of excitatory neurons
    t_events_I: times of events of inhibitory neurons
    rates_E: rates of excitatory neurons
    rates_I: rates of inhibitory neurons
    """
    n_E = n_I = n = 0
    t_events_E = [0]
    t_events_I = [0]
    rates_E = [mu_E]
    rates_I = [mu_I]
    time = [0]
    while n <= n_max:
        # Excitation neurons
        l_Enew = rates_E[-1]  + dt * (mu_E- rates_E[-1])/tau
        if np.random.uniform() < rates_E[-1]*dt:
            l_Enew += n_EE
            t_events_E.append(time[-1]+dt*np.random.uniform())
            n_E += 1
        if np.random.uniform() < rates_I[-1]*dt:
            l_Enew -= n_IE
            t_events_E.append(time[-1]+dt*np.random.uniform())
            n_E += 1

        # Inhibition neurons
        l_Inew = rates_I[-1] + dt * (mu_I- rates_I[-1])/tau
        if np.random.uniform() < rates_E[-1]*dt:
            l_Inew += n_EI
            t_events_I.append(time[-1]+dt*np.random.uniform())
            n_I += 1
        if np.random.uniform() < rates_I[-1]*dt:
            l_Inew -= n_II
            t_events_I.append(time[-1]+dt*np.random.uniform())
            n_I += 1
        rates_E.append(l_Enew)
        rates_I.append(l_Inew)
```

```
        time.append(time[-1]+dt)

        n = n_E + n_I
    return time, t_events_E, t_events_I, rates_E, rates_I

def identify_clusters_model(times, delta):
    """
    Identifies clusters in a temporal series given a resolution parameter delta
    Computes the size and duration of clusters

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

    ## Output:
    clusters: list of clusters
    clusters_sizes: list of sizes of clusters
    clusters_times: list of durations 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 = []

    clusters_sizes = [len(cluster) for cluster in clusters]
    clusters_times = [cluster[-1] - cluster[0] for cluster in clusters]
    return clusters, clusters_sizes, clusters_times

def bivariate_algorithm(rate1, rate2, muE, muI, nEE, nII, nEI, nIE):
    """
    Algorithm that computes interevent times and Hawkes intensity for a bivariate Hawkes
    process

    #Inputs:
    rate1: Previous excitation rate
    rate2: Previous inhibition rate
    nEE: "Strength" of the autoexcitation process
    nII: "Strength" of the autoinhibition process
    nEI: "Strength" of the excitation to the inhibition
    nIE: "Strength" of the inhibition to the excitation
    muE: Background intensity of the excitation
    muI: Background intensity of the inhibition


    #Output: ratex_k, x_k, reaction (0 for excitatory events and 1 for inhibitory events)
    """
    _, xk1 = algorithm(rate1, muE, nEE)
    _, xk2 = algorithm(rate2, muI, nII)

    xks = [xk1, xk2]

    if xk1 < 0:
        print(xk1)
        xk1 = 0
    if xk2 < 0:
```

```
        print(xk2)
        xk2 = 0

    reaction = np.argmin(xks)

    if reaction == 0:
        rate1_tk = (rate1 - muE) * np.exp(-xk1) + nEE + muE
        rate2_tk = (rate2 - muI) * np.exp(-xk1) + nEI + muI
    else:
        rate1_tk = (rate1 - muE) * np.exp(-xk2) + nIE + muE
        rate2_tk = (rate2 - muI) * np.exp(-xk2) + nII + muI

    if rate1_tk <= muE:
        rate1_tk = muE
    if rate2_tk <= muI:
        rate2_tk = muI

    xk = xks[reaction]

    return rate1_tk, rate2_tk, xk, reaction

def generate_series_bivariate(K, nEE, nII, nEI, nIE, muE, muI):
    """
    Generates temporal series for K bivariate Hawkes processes

    ##Inputs:
    K: Number of events
    nEE: "Strength" of the autoexcitation process
    nII: "Strength" of the autoinhibition process
    nEI: "Strength" of the excitation to the inhibition
    nIE: "Strength" of the inhibition to the excitation
    muE: Background intensity of the excitation
    muI: Background intensity of the inhibition

    ##Output:
    times_between_events: time series the interevent times
    times: time series the events
    rate1: time series for the intensity of process 1 (Excitation)
    rate2: time series for the intensity of process 2 (Inhibition)
    reactions: list the event type (0 for excitation. 1 for inhibition)
    """
    times_between_events = [0]
    rate1 = [muE]
    rate2 = [muI]
    reactions= []
    for _ in range(K):
        rate1_tk, rate2_tk, xk, reaction = bivariate_algorithm(rate1[-1], rate2[-1], muE,
 muI, nEE, nII, nEI, nIE)
        rate1.append(rate1_tk)
        rate2.append(rate2_tk)
        reactions.append(reaction)
        times_between_events.append(xk)
    times = np.cumsum(times_between_events)
    return  times_between_events, times, rate1, rate2, reactions
```