Efficient simulation of point processes with applications to neurosciences

Cyrille Mascart

UNIVERSITÉ CÔTE D'AZUR

ÉCOLE DOCTORALE
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

THÈSE DE DOCTORAT

Efficient simulation of point processes with applications to neurosciences

Cyrille MASCART

Laboratoire d'Informatique, Signaux et Systèmes de Sophia Antipolis

Présentée en vue de l'obtention du grade de docteur en informatique **d'**Université Côte d'Azur
Dirigée par : Alexandre Muzy
et co-dirigée par : Patricia Reynaud-Bouret
Soutenue le : 29/11/2021

Devant le jury composé de

Alexandre MUZY	Supervisor	CNRS, Université Nice Côte d'Azur
Patricia REYNAUD-BOURET	Co-supervisor	LJAD, Université Nice Côte d'Azur
Ryota KOBAYASHI	Reviewer	GSFS, The University of Tokyo
Ernesto KOFMAN	Reviewer	Departamento de Control, FCEIA
Eva LÖCHERBACH	Reviewer	SAMM, Université Paris 1
David HILL	President	LIMOS, Université Clermont Auvergne

Simulation efficace de processus ponctuels avec applications aux neurosciences

Efficient simulation of point processes with applications to neurosciences

Membres du jury:

Rapporteurs

Rapporteur	Ryota Kobayashi	Associate professor	GSFS, The University of Tokyo
Rapporteur	Ernesto Kofman	Professor	Departamento de Control, FCEIA
Rapportrice	Eva Löcherbach	Professeure	SAMM, Université Paris 1
Directeur	Alexandre Muzy	Chargé de recherche CNRS	CNRS, Université Nice Côte d'Azur
Co-directrice	Patricia Reynaud-Bouret	Directrice de recherche	LJAD, Université Nice Côte d'Azur
Président	David Hill	Professeur	LIMOS, Université Clermont Auvergne

Abstract

Stochastic and point processes are often used to model networks of spiking neurons. However, the number of neurons, even in a small mammal brain, is at least a few millions. There is therefore a strong need for efficient simulation algorithms. Nevertheless, traditional algorithms for point process simulation cannot simulate such large networks within a reasonable time and memory constraints.

In this thesis, we introduce new simulation algorithms for large networks of point and stochastic processes. By using the time asynchrony of point processes, and discrete-event techniques from computer science, we managed to reduce the time complexity of the simulation algorithm from $\mathcal{O}(M^2 \log M)$ to $\mathcal{O}(M \log M)$, where M is the size of the network. The algorithm was successfully applied to reduce the execution time of large networks of Ornstein-Uhlenbeck processes and Hawkes processes.

The new algorithm also displays a reduced memory complexity, from $\mathcal{O}(M^2)$ to $\mathcal{O}(M)$. However, the data structures for storing the connectivity matrices usually have a $\mathcal{O}(M^2)$ memory complexity. We proposed a new data structure for storing the connectivity matrix, based on its partial reconstruction when the simulation software needs access to one of the connections. This procedural connectivity is ensured by storing the internal state of a pseudo-random number generator, used to create the random connection matrix, instead of the connection matrix.

With such a small imprint, our algorithm is able to simulate a network composed of millions of Hawkes processes within minutes, on a single core laptop computer, paving the way for easier study of brain-sized structures, and in particular functional connectivity inference.

Keywords: stochastic simulation, exact simulation, large-scale simulation, point process, stochastic process, marked Hawkes process, Ornstein-Uhlenbeck process, discrete-event, spiking neurons, procedural connectivity, DEVS

Résumé

Les processus stochastiques et ponctuels sont souvent utilisés pour modéliser les réseaux de neurones à impulsions. Cependant, le nombre de neurones, même dans le cerveau d'un petit mammifère, est d'au moins quelques millions. Il y a donc un fort besoin d'algorithmes et de logiciels de simulation efficaces. Néanmoins, les algorithmes traditionnels de simulation de processus ponctuels ne peuvent pas simuler de si grands réseaux dans un temps et avec des contraintes de mémoire raisonnables.

Dans cette thèse, nous introduisons de nouveaux algorithmes de simulation pour les grands réseaux de processus ponctuels et stochastiques. En utilisant l'asynchronisme temporel des processus ponctuels et des techniques issues du monde de la simulation à événements discrets, nous avons réussi à réduire la complexité temporelle de l'algorithme de simulation de $\mathcal{O}(M^2 \log M)$ à $\mathcal{O}(M \log M)$, où M est la taille du réseau. L'algorithme a été appliqué avec succès pour réduire le temps d'exécution de grands réseaux de processus d'Ornstein-Uhlenbeck et de processus de Hawkes. Le nouvel algorithme présente également une complexité mémoire réduite, passant de $\mathcal{O}(M^2)$ à $\mathcal{O}(M)$. Cependant, les structures de données permettant de stocker les matrices de connectivité ont généralement une complexité mémoire de $\mathcal{O}(M^2)$. Nous avons proposé une nouvelle structure de données pour stocker la matrice de connectivité, basée sur la reconstruction partielle de la matrice de connexion lorsque le logiciel de simulation a besoin d'accéder à l'une des connexions.

Avec une empreinte aussi faible, notre algorithme est capable de simuler un réseau composé de millions de processus de Hawkes en quelques minutes, sur un ordinateur portable à un seul cœur, ce qui ouvre la voie à une étude plus facile des structures cérébrales, et en particulier à l'inférence de la connectivité fonctionnelle.

Mots clef: simulation stochastique, simulation exacte, simulation à grande échelle, processus ponctuel, processus stochastiques, processus de Hawkes marqué, processus d'Ornstein-Uhlenbeck, événement-discret, neurones impulsionels, connectivité procédurale, DEVS

Remerciements

Cette thèse n'aurait jamais pu aboutir sans mes directeurs de thèse, Alexandre et Patricia, et c'est tout naturellement à eux que vont mes premiers mots. Vous m'avez poussé (et support)é pendant 5 années, et je vous remercie de m'avoir permis de mener ce premier projet à bout malgré les difficultés.

J'en profite aussi pour remercier mes rapporteurs, Ryota Kobayashi, Ernesto Kofman et Éva Löcherbach. Vous avez été disponible malgré les diverses simagrés administratives, et vos retours me seront précieux pour améliorer mon travail à l'avenir. I would also like to take this opportunity to thank my reviewers, Ryota Kobayashi, Ernesto Kofman and Éva Löcherbach. You have been available despite the various administrative hassles, and your feedback will be invaluable in improving my work in the future.

On l'oublie trop souvent, mais la science est un travail de collectif et humain, et le travail qui suis est un bon exemple. Merci, Benny, de m'avoir mis en contact avec Alexandre, alors que je parlais doucement de m'intéresser aux neurosciences en école d'ingénieur. Merci Gilles (Scarella) pour ta présence, ton aide technique et nos discussions, pour avoir pu partager nos ressentis sur le travail que l'on faisait.

Merci, Oussama, pour ta présence, nos discussions, nos voyages et partage. Tu as été une source de chaleur inépuisable, à la fois au bureau et en-dehors. Merci pour nos partages, nos longues discussions scientifiques et philosophiques. J'ai hâte de pouvoir être de l'autre côté de la page en lisant ton manuscrit.

Thank you Yuri, you came from the other side of the world, for the better or the worst. You certainly added some better to my life.

Merci à tous les membres de l'I3S. Aux passagers temporaires, Benjamin, Émilien, Ophélie, Justin, Thibault, Johnatan, Déborah, Læticia, et tant d'autres. Merci pour les échanges, les fous rires. Merci aux permanents aussi, Gilles (Bernot), Jean-Paul, et d'autres, pour votre aide et votre présence rassurante.

Merci à vous, Estelle et Gaëtan, de m'avoir ouvert les portes de l'amitié, pour votre accueil. Pour les jeux, pour les discussions, pour toutes les découvertes et les partages. Vous avez définitivement changé ma vie, pour le meilleur, et je vous en suis infiniment reconnaissant.

Merci à toi Guilhem. Pour nos discussions interminables au téléphone, et ailleurs. Je te souhaite que tes souhaits se réalisent, aussi dur que cela puisse paraître.

Merci, Antoine, pour toutes ces soirées et ces jeux. Merci Alexandre (Bardakkof) pour nos débats, et de m'avoir donné souvent un autre point de vue sur le monde.

Merci Damien, Lénaïc, Lucien et Ben, pour tous ces moments ensemble, les rires. Merci au club de robotique, aux élèves et professeurs de l'ISIMA, de Harbin. Merci de m'avoir accompagné pendant ma formation.

Un merci général aux autres amis, les anciens, de Clermont-Ferrand, de Nice et d'ailleurs (ma mémoire flanche déjà, c'est terrible), vous avez fait partie de mon parcours, et vous avez certainement participé à ce que je suis devenu. Les rencontres sont comme les repas après tout, si l'on ne se souvient pas de toutes, ce sont bien elles qui nous composent.

Grazie a Dario, Eugenia e Primo, Grazie a Dario, per questi spuntini e momenti insieme. Che possano continuare sempre.

Grazie a tutta la famiglia Mezzadri, Antonella, Fabio, Gabrielle e Matilde (e Titta), per la vostra accoglienza e il vostro calore.

Un merci particulier à tous ceux qui, partis trop tôt, ne peuvent malheureusement pas voir la fin de cette partie de ma vie. Un ringraziamento speciale a tutti coloro che, essendo partiti troppo presto, purtroppo non possono vedere la fine di questa parte della mia vita.

Merci à ma première famille, Maman, Papa, Julie et Émilie. Vous m'avez toujours soutenu dans mes projets et êtes un point de repère dans ma vie.

E alla fine, grazie infinite a mio amore, la luce delle mie giorne. Merci pour nos longues discussions, sur tout et sur rien. Thank you for making me dream about a future with you, about new ideas. Grazie per avermi ispirato a diventare una versione migliore di me stesso. Merci d'avoir partagé avec moi tes peines et tes joies, et de partager les miennes. Thank you so much for Al mio sole perzonale. Grazie Giulia di essere, simplemente. Ti amo tanto.

Enfin un merci rapide aux arbres abattus pour créer tout le papier utilisé pendant mes études. J'espère que leur sacrifice en valait la peine.

List of Figures

Contents

1 Introduction

The human brain is a complex system: a network of hundreds of billions of neurons interacting by means of self-generated electrochemical currents, called action potential. Neurons can be viewed as signal processing devices [56, 73] or computing units, in the sense given by [150]: they aggregate these electrochemical input currents over time and send a response [38, 124] to the network once a certain threshold is reached. Even though these action potentials can have diverse and complex shapes ([38, 124]), they are usually abstracted to a single temporal point [59, 83, 91], as the information they carry is mostly contained in the emission time of the signal.

The mathematical framework which can deal with sporadic information is called *point process theory*. The origin of this theory can be traced to the origins of probability theory itself, with life tables modeling [28]. They are now used in a large variety of fields, such as (but not exhaustively):

- In chemistry, point processes model chemical transformation [41]
- In finance, point processes model market stock price changes [4, 8, 21]
- In geology, point processes model earthquakes and basaltic area distribution [25, 107, 121]
- In physics, point processes model the disintegration time of radioactive atoms [115]
- In demography, point processes model population size changes [68, 121]
- In biology and medicine, point processes model groups of patients [92], epidemic infections [121, 143], gene regulation [46]

Finally, in neuroscience, point processes can be used to model neuronal current emission [63].

Most applications of point processes seldom use networks of point processes with sizes bigger than a dozen nodes. In medicine, Mancini [92] uses a marked Hawkes point process composed of four marks (here groups of patients). In the context of genetics, Hansen [46] uses a marked point process with up to eight marks (here transcription factors) in the three papers presented in her thesis. Hawkes point process models in finance typically use less than 10 marks (see for instance Bacry's review [4]). Vestergaard simulated a contagion process with up to 1000 nodes

in an epidemiological study [143]. However, particle physics and chemistry are also interested in large scale simulations, see for instance [12].

The research in algorithms for point processes reflects this state of fact and are unable to simulate very large networks of point processes. This is an issue for creating and validating new models and methods involving networks of millions or billions of point processes. Such algorithms could become useful for many applications of point processes, such as crime modeling [121], epidemiology [121] and, of course, neuroscience [63, 117].

This thesis deals with the improvements of current simulation algorithms for point processes, with applications to neuroscience where the number of point processes to simulate is large. In this introduction, we will first precise and disambiguate some vocabulary that have different meanings across the three fields this thesis is built upon, namely mathematics, computer science and biology. In a second time, we will present the point process theory in more details, before giving a brief overview of the biological background. This introductory chapter ends with a review of the computer science theoretical framework behind the proposed algorithms and implementations.

1.1 Vocabulary

Different communities tend to have diverging vocabularies. Scientific communities do not escape this fate, and transdisciplinary work is doomed to use vocabularies that might be misunderstood, depending on the discipline on which the reader works. We collect here, at the beginning of the introduction for easiness of retrieval, the vocabulary that might be misunderstood whether the reader adopts the point of view of a mathematician or a computer scientist.

Time asynchrony

In computer science, the term *time asynchrony* refers to the hypothesis that, in a system composed of many parts communicating with each other (for instance a network of neurons), only one new event can happen in the system at any time.

In *point process theory*, the same hypothesis exists, and a process for which, at any time t, there can be only one occurring point is called a *simple process*. When the conditional intensity exists for a point process, this hypothesis is automatically verified by construction. The expression *time asynchrony hypothesis* is used, for instance in chapters 3 and 4 instead of *simple process*.

Event

An event is "something" that can happen. In probability theory, an event is a set to which a probability measure can be assigned [13].

Figure 1.1: A graphical illustration of the realization of a point process (here a unit rate Poisson point process), on the interval [0, 10]. Each triangle represents an occurrence of the point process.

In programming, an event is any action or occurrence that can be recognized by the computer [67, 144], for instance an interruption of the processor by an Input/Output (I/O) interrupt message from the hard drive or the keyboard.

In discrete event systems, a discrete event is a moment in time assigned to an instantaneous possible change of state of a system [6, 118, 154].

Process

In mathematics, a process is a function.

In computer science, the term process often refers to an instance of a computer program run by the operating system (OS). In simulation modelling, a process is a sequence of activities, or a sequence of events (an activity is what transforms an object, it begins and ends with an event) [6].

For agent-based models, a process is also the other name given to an agent.

1.2 Mathematical background

Probability basics

We refer to the appendix A1 of [13] for a reminder of the basics of probability theory.

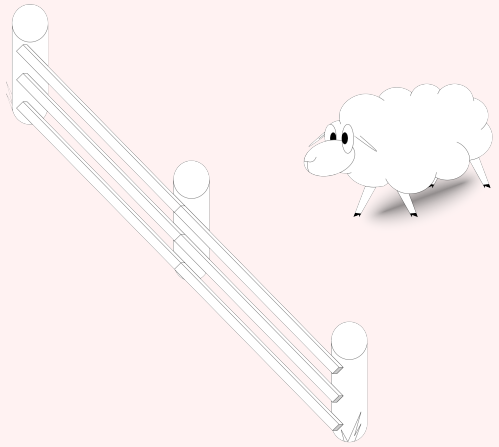
Point Process

Multiple definitions of point processes have been developed [13, 28, 126]. Given a continuum χ, a *point process* $\mathcal{N}$ is a random countable subset of points $\mathcal{N} = \{X_i\}_{i=1,\dots,M, M\in\mathbb{D}}$ on the continuum χ. Some examples have been given at the beginning of this introduction and even more can be found in [138]. From now on we will restrain our analysis to point processes on the set of positive real numbers $\chi = \mathbb{R}_+ = [0, \infty[$ for simplicity, but definitions and some results can be extended to any continuum, the plane $\chi = \mathbb{R}_+ \times \mathbb{R}_+$ for instance [5]. Point processes on the real line are often called temporal point processes [120, 138].

Let $\mathcal{N}$ be a temporal point process on $\mathbb{R}_+$. We call a *realization* $\omega = \{t_1, \dots\}$ of the point process $\mathcal{N}$ a countable subset of $\mathbb{R}_+$ that is also a possible outcome of the point process $\mathcal{N}$. The figure 1.1 is a graphical representation of the realization of a point process, where each triangle represents a point.

Red string example

It is time to introduce the red string example of this section, which will appear in boxes like the current one. This is the story of a statistician, struggling to get asleep. Like any human beings in their situation would do, (s)he decides to imagine sheep jumping over a fence. Unlike most human beings, the statistician needs to imagine the time when a sheep jumps as a realization of a temporal point process. We will try to characterize the possible point process $\mathscr{S}$ imagined by our sleepy, yet impish, statistician.



There are several possible approaches to characterize point processes ([28] enumerates four), the main one consists of characterizing the distribution of the points on the continuum space $\mathcal{X}$. For instance, one could look at the interpoint interval distribution. The case where the interpoint intervals are independent and identically distributed (*iid*), and exponentially distributed is called a Poisson point process, which is one of the most commonly used point processes. Here we will start with another approach, and consider a characterization of some point processes called the conditional intensity function. But first, we must introduce some important concepts.

1.2.1 Counting process

A *counting process* [13, 36] is a function (here a function of time, N_t) associated to a point process $\mathscr{N}$. Informally, the counting process "counts" the number of points T_i from the point process $\mathscr{N}$ that happened before t. Let A be a subset of $\mathbb{R}_+$. The counting process N_t is the number of points of $\mathscr{N}$ lying in A, then $\forall t \in \mathbb{R}_+$:

$$N_t = \sum_{T^i < \sup A} \delta_A(T^i), \quad \text{with } \delta_A(T^i) = \begin{cases} 1, T^i \in A \\ 0, \text{otherwise} \end{cases}.$$

Red string example

The definition is explicit: the counting process associated to the point process $\mathscr{S}$ is the function $t \to S_t$ counting the number of sheep that have jumped over the fence in the time interval $[0, t]$.

Using this definition, it is easy to understand that $\forall (t, t') \in \mathbb{R}_+^2, t' > t$, the value $N_{t'} - N_t$ is the number of points of the point process $\mathscr{N}$ lying in $[t, t']$. Following from this remark, we define the infinitesimal notation of the counting process as $\mathrm{d}N_t = N_t - N_{t-\mathrm{d}t}$. For sufficiently small $\mathrm{d}t$, $\mathrm{d}N_t$ is a Dirac comb.

1.2.2 Filtration

Informally, the *filtration*, or *history*, $\mathscr{H}_t$ at time t, associated to a point process $\mathscr{N}$, is the total information needed for simulating the point process at time $t' \geq t$. Mathematically, $\mathscr{H}_t$ is a sub-sigma-algebra of the total set of possible events $\mathscr{F}$ [13]. $\mathscr{H}_t$ is composed of the possible events happening before time t. The mathematical details are not necessary for understanding what follows, so we refer the inquisitive readers to the first chapter of [13]. The concept is, however, important to define the conditional intensity of a point process.

Red string example

For our simple point process, the filtration $\mathscr{H}_t$ at time t contains, in particular, all the points strictly before t. In other words, this corresponds to all the times when a sheep has jumped over the fence.

1.2.3 Stationarity

A point process is said to be stationary if the distribution of the point process in any interval is invariant by translation of the interval. Said otherwise, the distribution of the points on any interval depends on the length and not the location of the interval, that is

$$\mathbb{P}N_{[t,t+x]} = k, \quad x > 0, k = 0, 1, \ldots$$

depends on x and not on t.

The stationarity of the point process is most of the time a given hypothesis, but it is important because the demonstration of most properties of point processes depend on it. Precise mathematical insights can be found in [14], notably the demonstration of the convergence to stationarity of

the Hawkes process, under certain conditions. Some algorithms are made so that the points are computed by going back in time, which is a way to avoid this issue. See [117] for instance.

Note that the distribution of the first point of a simulated point process is often not stationary, because the simulation of a point process generally starts with an empty past. Therefore, the probability of having a certain realization at the beginning is clearly not invariant by translation. Bremaud [14] demonstrates convergence towards the stationarity for bounded memory processes.

1.2.4 Intensity function

The (conditional) *intensity function* $\lambda(t|\mathscr{H}_t)$ of a point process $\mathscr{N}$ is, informally, the instantaneous rate of occurrence of new points. The existence of the intensity function for any stationary temporal point process has first been shown by Khinchin [69]. The function is related to the expectation of a point occurrence in an infinitesimal interval $[t, t + \mathrm{d}t[$ (see chapter 5 of [127]):

$$\mathbb{P}\, N_{[t,t+\mathrm{d}t[} = k|\mathscr{H}_t = \begin{cases} 1 - \lambda(t) + o(\mathrm{d}t), & k = 0 \\ \lambda(t) + o(\mathrm{d}t), & k = 1\,, \\ o(\mathrm{d}t), & k > 1 \end{cases}$$

where Landau's *little-o* notation $o(\cdot)$ is understood at the limit $\mathrm{d}t \to 0$: $f(t, \mathrm{d}t) = o(\mathrm{d}t) \iff \forall t, \forall c > 0, \exists k > 0, \forall 0 < \mathrm{d}t < k, 0 \leq c\, f(t, \mathrm{d}t) \leq \mathrm{d}t$. Said otherwise, $o(\mathrm{d}t)$ means "negligible with respect to $\mathrm{d}t$".

The intensity function, when it exists, fully characterizes the distribution of a point process. The conditional intensity function with the set of past points of the point process fully characterize the point process itself. For instance, a constant intensity function defines a homogeneous Poisson process. The particular case where $\lambda = 1$ is often called unit-rate Poisson process. The case $\lambda(t) = f(t)$, where the intensity depends solely on the time variable, is called inhomogeneous Poisson process, and is, of course, the generalization of the homogeneous case.

Red string example

As a first approximation, we will make the hypothesis that the point process dreamed by our statistician is a unit-rate homogeneous Poisson process. The random variable defined by the time between two consecutive points has an expected value of 1, meaning on average, *one* sheep per second are expected to jump over the fence. Sheep also do not influence each other in this model: observing a jump does not change the probability to observe another jump.

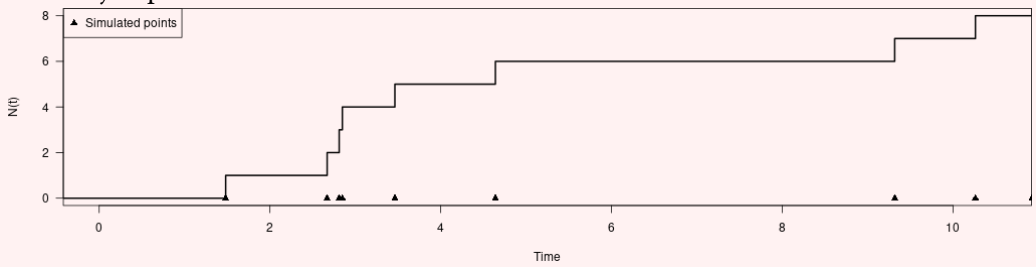


Figure 1.2: A realization of an homogeneous Poisson process, with parameter $\lambda = 1.2$, represented as triangles on the bottom. The realization of the counting process associated to this point process is represented as well.

In the more complex cases where the intensity function depends on the time t and on a filtration $\mathcal{H}_t$ of the point process at time t, the intensity function is called *conditional* intensity function (the contingency on the history is implied), and can be noted $\lambda(t, \mathcal{H}_t)$ or $\lambda(t|\mathcal{H}_t)$. We use the later notation to invite the reader to remember that, because of its dependence on the past of the point process, the conditional intensity function is itself a random variable.

Hawkes point process

The Hawkes process [48, 49] is a self-exciting process that is defined by its characteristic intensity function

$$\lambda(t|\mathcal{H}_t) = \nu + \int_0^t h(t-s)\,\mathrm{d}N_s = \nu + \sum_{T_k<t} h(t-T_k)$$

where ν is a constant value, called the background intensity, $h(t)$ is a self-excitation function, often $h(t) = \alpha \exp(-\beta t)$, for some $\alpha, \beta > 0$, and N_t is the counting process associated with the Hawkes point process.

The Hawkes process can be seen as an extension of the Poisson process, where ν is the intensity of an homogeneous Poisson process, and the kernel $h(t)$ is a self-excitation term. The conditional intensity presented here actually corresponds to the linear Hawkes process. Other versions exists, for instance as an extension of a nonhomogeneous Poisson process ($\nu \mapsto \nu(t)$). An important

modification is the non-linear Hawkes process, which is the result of the composition of the linear conditional intensity with another function ϕ: $\lambda(t|\mathcal{H}_t) = \phi\Big(\sum_{T_k<t} h(t-T_k)\Big)$ [14].

1.2.5 Multivariate (or marked) process

Until now we have been examining the case of univariate temporal point processes. This is the case where the coordinates of the points are temporal values. However the possibility to classify events into categories is a critical modeling tool. Point process theorists introduced the concept of multivariate point processes (see the founding article [26]).

In a multivariate (or marked) point process, the coordinates of the points are bidimensional in $(\mathbb{R} \times \mathbb{N})$: a type (or label), often chosen in a finite set of natural numbers $\mathbb{N}$, is attributed to all the points. In the following, the terms marked or multivariate, and type or label will be used without any further distinction. The point process generating the points without the marks is called the *ground process*, while the point process associated with a single mark is called a *marginal process*.

The previously introduced definitions and properties still hold, and we refer the inquisitive readers to the chapter 7.3 of [28]. The counting process associated with the ground point process is $N_t = \sum_{k=1}^{M} N_t^k$, where each marginal process $\mathcal{N}^k, k \in \{1, \dots, M\}$ is counted by the counting process N_t^k. For any given k, $\mathcal{H}_t$ is the history of the ground process at time t, and $\lambda^k(t|\mathcal{H}_t)$ the conditional intensity function of the marginal process k. The ground point process has intensity $\lambda(t|\mathcal{H}_t) = \sum_{k=1}^{K} \lambda^k(t|\mathcal{H}_t)$. Note that the history of any marginal point process is the history of the ground process itself. Informally, the information needed to simulate the marginal process k may indeed be contained in the history of the other marginal processes.

Red string example

For instance, a sheep's fur can be either gold or white, but there are fewer golden ones than white ones (10% vs. 90%). The ground process is the point process whose realization is the time of jumps, blind to the color of the jumping sheep. At the moment, let us consider that the ground process is the same process generating the time of jumps as before (unit rate Poisson process). There are two marginal processes here: one generating the points with mark 'golden' (we will call it G, with intensity λ^G), the other one for the points marked 'white' (called W, with intensity λ^W). Combining two Poisson processes X and Y with respective intensity μ and ν results in a ground Poisson process with intensity $\lambda = \lambda^X + \lambda^Y = \mu + \nu$. So here it can be assumed $\lambda^G = 0.1$ and $\lambda^W = 0.9$.

Multivariate Hawkes process

For the multivariate Hawkes process composed of M marginal processes, the intensity function of any marginal process i is

$$\lambda^i(t) = \nu + \sum_{j=1}^{M} \int_0^t h^{j\to i}(t - s^j)\, \mathrm{d}N^j_{s^j} = \nu + \sum_{j=1}^{M} \sum_{T^j_k < t} h^{j\to i}(t - T^j_k)$$

For each couple of processes (i, j), the function $h^{j\to i}(t)$ is either $h^{j\to i}(t) = 0$ or a function of time.

In the case of the Hawkes point process, univariate or multivariate, a sufficient condition for stability is given by the value of the spectral radius of the kernel, which must be strictly less than 1. In other words, for a Hawkes process composed of M marginal processes:

Theorem 1. *[48] Let $\lambda^i(t|\mathscr{H}_t) = \nu^i + \sum_{j=1}^{M} \int_0^t h^{j\to i}(t - s)\, \mathrm{d}N_s$ be the intensity of the marginal process i. Let $\rho_1, \dots, \rho_M$, be the eigenvalues of the matrix $\mathrm{K} = \left(\int_0^\infty h^{j\to i}(s)\, \mathrm{d}s\right)_{(i,j)\in\{1,\dots,M\}^2}$. The spectral radius of K is defined by $\rho(\mathrm{K}) = \max\{|\rho_1|, \dots, |\rho_M|\}$. Then the ground process is stable if $0 < \rho(\mathrm{K}) < 1$.*

Note that in the case of the univariate Hawkes point process, the condition simply means that the integral on the half-line $[0, \infty[$ of the kernel $h(t)$ must be strictly less than 1 in absolute value. Using the common choice $h(t) = \alpha \exp(-\beta t)$, $\alpha, \beta > 0$, the condition translates into: the process is nonexplosive if $\frac{\alpha}{\beta} < 1$. The condition can then be understood, in plain English, as: "If the function $h(t)$ does not take too high values during too long period of time, then the process is nonexplosive".

Local independence and graphical representation

Red string example

In sheep society, color is everything. A golden sheep tends to attract other ovines by natural leadership, hence when a golden sheep jumps over, more white sheep jump too. White sheep also influence each other directly, a white sheep alters the judgment of their kind so that they act alike, though with less efficiency than golden sheep. Golden sheep, though, tend not to mind their surroundings: their own beauty is so distracting they often cannot think about anything else but themselves.

Developed by [133] for Markov processes, and extended to stochastic processes by [33, 34], the concept of local-independence defines the local, temporal dependencies between the processes. It is based on an extension of Granger-causality, which is a simple and direct form of causality.

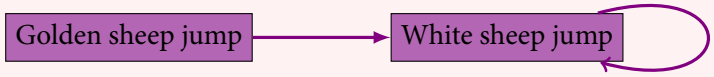
Informally, let X_t, Y_t be variables of time, the goal being to forecast X_{t+1}. If Y_t always precedes X_{t+1}, and Y_t contains information useful for predicting X_{t+1} that cannot be found in other variables, then Y_t is said to Granger-cause X_{t+1} (see for instance [134] for more mathematically detailed explanation and some personal accounts about the development of Granger-causality by Professor Clive Granger, which inspired this informal explanation).

The notion of Granger-causality is important here, and is embedded in the "locally" term: local independence from i to j does not mean the marginal processes i and j are independent. Indeed, there can still be dependence from j to i, or, for instance, there can be a third process h on which i and j both depend. An extreme case is when there is an interaction from process i to process h to process j. The process j is Granger-independent from the process i, though the activity depends indirectly on the activity of i.

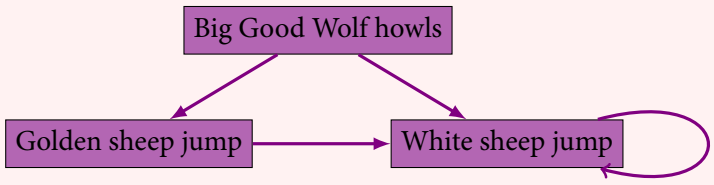
The local-independence property translates perfectly in terms of graphical representation. For self-exciting processes, the local-independence graph represents the interaction between processes: in a graph where the vertices are the marginal processes, the absence of a directed edge from process j to i represents the absence of direct interaction from process j to process i. Note that the edges are directed, so the graph can have an edge $i \to j$ and no edge $j \to i$.

Red string example

The graphical representation of the local-independence graph in our example looks like:



The cattle hear the rumor of wolves in the nearby, and we can wonder: how will the sheep react? Nothing to worry about, they aren't dangerous in this imaginary world, but our sheep generally prefer not taking risk. This will modify our independence graph, however. Do you see how, dear reader?



These concepts may look simple but are the core of the algorithms developed in chapters 2 and 3.

Explosivity of the marked point process

The realization of a point process is said nonexplosive if the limit of the sequence of points $(T_n)_n$

$$T_n < +\infty \implies T_n < T_{n+1}$$

describing the point process is infinite [28]:

$$T_\infty = \lim_{n\to\infty} T_n = +\infty$$

Deciding whether a point process is explosive given a set of parameters is often a difficult problem. However, most results on point processes hold only for nonexplosive realizations of the point process. Homogeneous Poisson processes with intensity $\lambda < \infty$ can never be explosive, since the number of points on any interval $[A, B]$ follows a Poisson distribution with parameter $\lambda|B - A|$. For marked point processes, the interactions between the processes can break the nonexplosivity hypothesis. The nonexplosivity can be guaranteed a.s. by constraining the parameters of the point process. This can, however, prove difficult for very large networks, like in chapter 4.

1.2.6 Martingales

The term martingale refers to a gambling strategy in 18$^{\text{th}}$-century France: to double the stake at each loss, which guarantees a positive gain, provided enough money and at least a winning bet. The exact origin of the word still remains unclear (read [93] for a thorough review of the possible etymologies), but the concept has become one of the foundation stones of the theory of stochastic processes.

Definition 1.2.1 (Martingale, from appendix A3.4 of [28])**.** If we denote $(\Omega, \mathscr{F}, \mathbb{P})$ a probability space, and assuming $\mathscr{H}$ a history on $(\Omega, \mathscr{F})$ and the real-valued process $X(\cdot) = \{X(t) : 0 \leq t < \infty\}$, X is a $\mathbb{P}$-$\mathscr{H}$-martingale if for $0 \leq s < t < \infty$, $\mathbb{E}[X(t)|\mathscr{H}_s] = X(s)$.

In other words, a martingale is a stochastic process whose expected value at time t, though conditioned on a certain history at t $\mathscr{H}_t$, depends solely on its last value $X(t)$.
The intensity function has another property, which is the most often used in practice. Indeed, when the intensity function exists, it is possible to create a martingale from the counting process N_t and the intensity function $\lambda(t)$ associated to a point process $\mathscr{N}$ (see lemma 7.2.V of [28]):

$$M_t = N_t - \int_0^t \lambda(s)\mathrm{d}s \text{ is a martingale.}$$

The function $\Lambda(t) = \int_0^t \lambda(s)\mathrm{d}s$ is called the compensator of the counting point process. This function is also critical in the formalization of the random-time change theorem, a fundamental piece of the point process theory stating that any temporal point process with conditional intensity $\lambda(t)$ can be reverted to a unit-rate Poisson process by a simple stretch of time $\tau(t) = \Lambda(t)$.

1.2.7 Random-time change theorem

Though the theorem has been formally demonstrated in 1974 by Papangelou [112], it has been in the mind of point process theorists for a longer time (see Watanabe's article [145], who is the first to have characterized the Poisson process by the form of its compensator $\Lambda(t)$ in 1964, and 1971 Meyer's article [99]). We will give the readers a simple form of the theorem, and as always refer the curious ones to the chapter 7 of [28] and 14 of [29]. The name of the theorem also changes depending on the author. For instance, Brown *et al.* call it the time-rescaling theorem [18, 47].

Theorem 2. *From [29] Let $\mathcal{N}$ be a simple point process adapted to the history $\mathcal{H}$ with bounded, strictly positive conditional $\mathcal{H}$-intensity $\lambda^*(t)$ and $\mathcal{H}$-compensator $\Lambda^*(t) = \int_0^t \lambda^*(s)\,\mathrm{d}s$ that is not a.s.-surely bounded. Under the random time change $t \to \Lambda^*(t)$, the transformed process*

$$\overline{N}_{\Lambda^*(t)} = N_t$$

is a Poisson process with unit rate.

In simpler terms, if there has been a total of N_{t^-} events in the interval $[0, t[$, with $\{t_1, \dots, t_{N_{t^-}}\}$ the times of the events, then the values of the compensator on the points $\Lambda(t_k), \quad \forall k \in \{1, \dots, N_{t^-}\}, t_0 = 0$ are the realization of a unit homogeneous Poisson point process. A more practical form is

$$\forall k \in \{1, \dots, N_{t^-}\}, u_k \sim \mathcal{U}([0, 1[), t_0 = 0, \quad \Lambda(t_k) - \Lambda(t_k) \sim -\log(1 - u_k)$$

This theorem is often used for simulation and goodness-of-fit tests (see chapter 3).

1.2.8 Simulation

There are many methods for simulating point processes [12, 30, 85, 102, 106]. Many methods exploit specific properties of point processes to try to accelerate the simulation algorithm. We will see here some of the generic methods that have been developed for making a realization of a point process. An introduction to point process simulation is also present at the beginning of chapter 3, and chapters 2 and 3 both propose new algorithm ideas for point processes.

Simulation of a homogeneous Poisson process

The simulation of a homogeneous Poisson process of intensity λ is pretty straightforward, and is detailed in the algorithm 1. It is based on the properties of the distribution of the points of an homogeneous Poisson process.

Algorithm 1 Simulating an homogeneous Poisson process with intensity λ on $[A, B]$ in parallel

Require: λ, A, B
  Generate $p \sim \mathscr{P}(\lambda|B - A|)$ (Poisson distribution with parameter $\lambda|B - A|$)
  for all $k = 1 \dots p$ **do**
    Generate $t_k \sim U([A, B])$
  end for
  return $\{t_i\}_{i=1,\dots,p}$

Simulation by the inverse-method

Using the random time change theorem, the interpoint intervals are given by the successive observation of the random variables $T_n = T_{n-1} - \frac{\log U(]0,1])}{\lambda}$.

Algorithm 2 Simulate an homogeneous Poisson process with intensity λ on $[A, B]$

Require: λ, A, B
1: Initialize $t = A, n \leftarrow 0$
2: **while** $t < B$ **do**
3: Generate $u \sim U(]0, 1])$
4: Set $t = t + -\frac{\log u}{\lambda}$
5: Set $n = n + 1$
6: Let $t_n = t$
7: **end while**
8: **Return** $\{t_i\}_{i=1,\dots,n}$

The figure 1.2, illustrating the concepts of realization and counting process, has been generated using algorithm 2.

For simulating inhomogeneous Poisson processes, and other kinds of point processes, other methods have been introduced.

Thinning (Lewis and Shedler [85])

The thinning algorithm 3 was introduced in [85] for simulating, over $[A, B]$, $A, B > 0$, an inhomogeneous Poisson processes N with bounded intensity $\lambda(t)$. The algorithm requires the simulation

of another Poisson process N^* (homogeneous or not), with an intensity $\lambda^*(t)$, given the relation $\lambda(t) \leq \lambda^*(t)$ and a simulation interval $[A, B]$.

The condition line 10 of the algorithm 3 is what is called thinning. The name is explicit: Lewis and Shedler (see Theorem 1 of [85]) demonstrated that for simulating an inhomogeneous Poisson process with intensity $\lambda(t)$, another inhomogeneous Poisson process with dominating intensity $\lambda(t) \leq \lambda^*(t)$ can be simulated while the resulting points $\{T^k\}_{k=1,\dots}$ are deleted with probability $1 - \frac{\lambda(T^k)}{\lambda^*(T^k)}$.

Algorithm 3 Simulate a nonhomogeneous Poisson process with intensity $\lambda(t)$ on $[A, B]$

Require: $\lambda(t)$, $\lambda^*(t)$, A, B

1: Generate points of the Poisson process N^* with intensity function $\lambda^*(t)$.
2: Let n be the number of points generated
3: **if** $n = 0$ **then**
4: Exit, there are no points in the point process N in the interval $[A, B]$.
5: **else**
6: Denote $T_1^*, T_2^*, \dots, T_n^*$ the ordered points.
7: Set $i = 1$ and $k = 0$.
8: **repeat**
9: Generate $u \sim \mathscr{U}[0, 1]$.
10: **if** $u \leq \frac{\lambda(T_i^*)}{\lambda^*(T_i^*)}$ **then**
11: Set $k = k + 1$ and $T_k = T_i^*$.
12: **end if**
13: Set $i = i + 1$.
14: **until** $i = n$
15: **end if**
16: **return** $\{T_i\}_{i=1,\dots,k}$

The algorithm is quite generic, no particular assumption being made on the bounding inhomogeneous Poisson process. The algorithm becomes interesting when considering the case where the bounding process N^* is a homogeneous Poisson process, with constant intensity function $\lambda^*(t) = \lambda^*$. The algorithm 1 for simulating a homogeneous Poisson process can then be used to generate the points $T_1^*, T_2^*, \dots, T_n^*$.

In the case where the upper-bound λ^* is so far from the real intensity function $\lambda(t)$ that the condition line 10 of the algorithm 3 is almost always false, a piecewise constant function can be used instead. If we denote such function $\lambda^*(t) = \sum_{k=1,\dots,K} \lambda_k^* \mathbb{1}_{[a_k,b_k]}$, with $\mathbb{1}_.$ the indicator function, and $\{[a_k, b_k]\}_{k=1,\dots,K}$ a set of pairwise distinct intervals entirely covering $[A, B]$, then the algorithm simply becomes:

The algorithm suffers from the need for an absolute bound existing for all possible past histories of the point process. Finding such an absolutely dominating function can be difficult to achieve

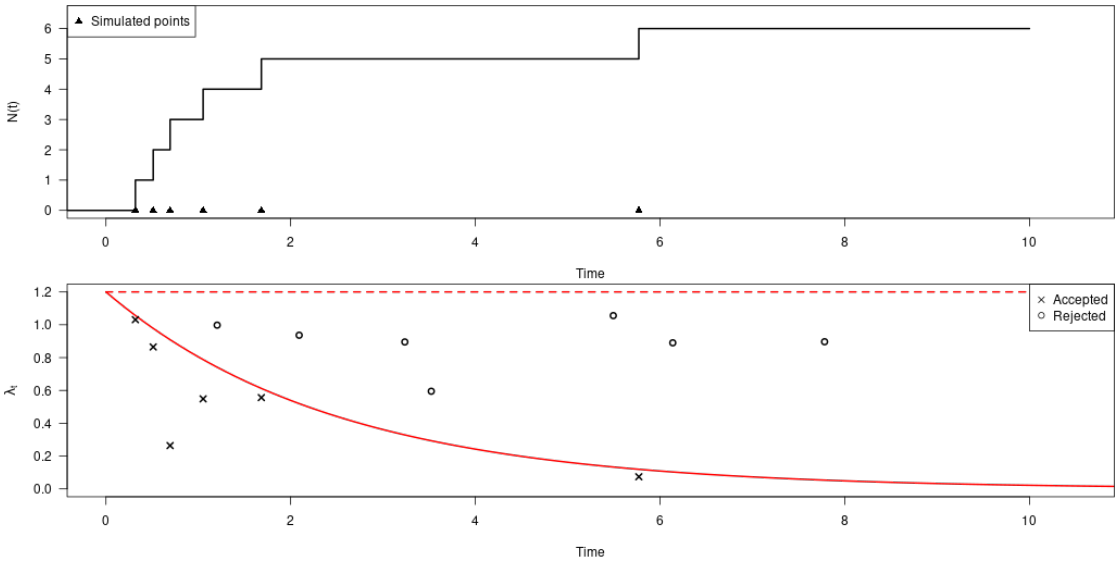


Figure 1.3: **Top**: A realization of an nonhomogeneous Poisson process N, with parameter $\lambda(t) = \alpha \exp(-\beta t)$, with $\alpha = 1.2, \beta = 0.4$, represented as triangles on the bottom of the counting process associated to N. **Bottom**: The intensity of the Poisson process is displayed, as well as the thinning procedure. The y-axis represents the uniform draws, and the red dashed horizontal line represents the homogeneous Poisson process N^* with intensity $\lambda^*(t) = \alpha$, which is used as dominating process. The uniformly distributed values $\lambda^* u \sim \mathscr{U}[0, \alpha]$ are displayed as 'o' (when the thinning rejected them) or 'x' (when they were accepted by the thinning procedure).

when the exact dynamic of the intensity function $\lambda(t)$ is hard to compute in advance, for instance when the analytical equation is unknown and the function is approximated *via* numerical analysis.

Modified thinning algorithm (Ogata [106])

The algorithm introduced by Ogata in [106] expands on the algorithm introduced by Lewis and Shedler (see section 1.2.8) for univariate inhomogeneous Poisson process. However, in contrast to the other algorithm, the modification introduced by Ogata allows the simulation of point processes whose intensities are random variables, such as for the Hawkes point process. The algorithm

Algorithm 4 Simulate a nonhomogeneous Poisson process with intensity $\lambda(t)$ on $[A, B] = \cup_{i=1,\dots,K}[a_i, b_i]$

Require: $\lambda(t), \{\lambda_k^*\}_{k=1,\dots,K}, \{[a_k, b_k]\}_{i=1,\dots,K}$
1: **for all** $k = 1, \dots, K$ **do**
2: Apply 1 with parameters $\lambda(t), \lambda_k^*, a_k, b_k$.
3: **end for**
4: **return** $\{T_i\}_{k=1,\dots,k}$

proposed by Ogata is also similar to another algorithm proposed by Gillespie [40, 41] for simulating stochastic systems with discrete states and where time advances with exponentially distributed jumps. The algorithm proposed by Gillespie being less generic than Ogata's modified thinning, we will detail only the latter.

The idea behind the algorithm is, if $\lambda(t)$ denotes the conditional intensity of the point process to simulate, to construct iteratively a piecewise constant function $\lambda(t)^*$ so that $\lambda(t) \leq \lambda^*(t)$ at all time. The algorithm is presented in algorithm 5. Note that in the original paper, two algorithms were presented, one for the case where $\overline{\lambda}(t)$ is a monotonically decreasing function of time (if no new points occur), and a second for a monotonically increasing function of time (again, without the addition of new points). We chose here to present the algorithm for the monotonically decreasing case, since this is the case that we are interested in for neuronal simulation (see chapter 3).

The algorithm 5 requires prior knowledge of the minimum value ν of the intensity function $\overline{\lambda}(t|\mathscr{H}_t)$ and of the maximum increment α of the intensity function $\overline{\lambda}(t|\mathscr{H}_t)$ at a new point. The lines 13 and 14 of algorithm 5 correspond to the step of choosing the mark of the point if the simulated process is a multivariate point process with marginal intensities $\lambda^i(t|\mathscr{H}_t)$, where $i = 1, \dots, M$ denotes the marks. In this case $\overline{\lambda}(t|\mathscr{H}_t) = \sum_{i=1}^{M} \lambda^i(t|\mathscr{H}_t)$. A proof of the validity of the procedure can be found in [106], page 24, Proposition 1. In the univariate case, these lines are omitted (since $M = 1$).

In the algorithm 5, we will note the filtration as a series of points $\{t_1, \dots, t_{n-1}\}$. This is a simplification of what the filtration really is, which can contain more than past points. This notation however will keep clearer when the conditional intensity function $\overline{\lambda(t|\mathscr{H}_t)}$ is updated after a new point has occurred. Therefore, if at time t all the past points of the point process are noted $t_1 < \dots < t_{n-1} < t$, then the conditional intensity function will be noted $\overline{\lambda}(t|t_1, \dots, t_{n-1})$. When $n = 0$ or $n = 1$, it is assumed that no points has occurred yet.

In Ogata's version of the algorithm, the conditional intensity $\overline{\lambda(t|t_1, \dots, t_{n-1})}$ is implicitely updated when n is increased, while we chose to write it explicitly here. Moreover, the new variable $p \in \{0, 1\}$ is here used to simplify the algorithm. In Ogata's version [106], a conditional jump is used depending on whether the new point is accepted or not by the thinning procedure line 10 of algorithm 5. Here the boolean p is set to 0 when the point is rejected, and 1 when the point is accepted.

The figure 1.4 illustrates the procedure in the particular case of the Hawkes point process.

Algorithm 5 Ogata's modified thinning algorithm, on $[0, T_{\max}]$

Require: $\overline{\lambda}(t|t_1, \dots, t_{n-1})$, $T_{\max}$, ν, α

Set $k = 0, t = t_0 = 0, \lambda^* = \nu, n = 0, p = 0$
loop
  Generate $u \sim \mathscr{U}[0, 1]$
  Set $\lambda^* = \overline{\lambda}(t|t_1, \dots, t_{n-1}) + p\alpha$ (λ^* is still an upper bound since $\overline{\lambda}(t|t_1, \dots, t_{n-1})$ is supposed to be monotonically decreasing)
  Compute $t = t - \frac{\log u}{\lambda^*}$
  if $t > T_{\max}$ **then**
    Stop; there are no new points to generate
  end if
  Generate $v \sim \mathscr{U}[0, 1]$
  if $v \leq \frac{\overline{\lambda}(t|t_1, \dots, t_{n-1})}{\lambda^*}$ **then**
    Set $n = n + 1, t_n = t, p = 1$ (a new point has occurred)
    Update the conditional intensity $\overline{\lambda}(t|t_1, \dots, t_n)$
    Generate $v \sim \mathscr{U}[0, 1]$
    Compute the mark i so that $\sum_{m=1}^{M-1} \lambda(t|\mathscr{H}_t)^m < v\overline{\lambda}(t) \leq \sum_{m=1}^{M} \lambda(t|\mathscr{H}_t)^m$
  else
    Set $p = 0$
  end if
end loop

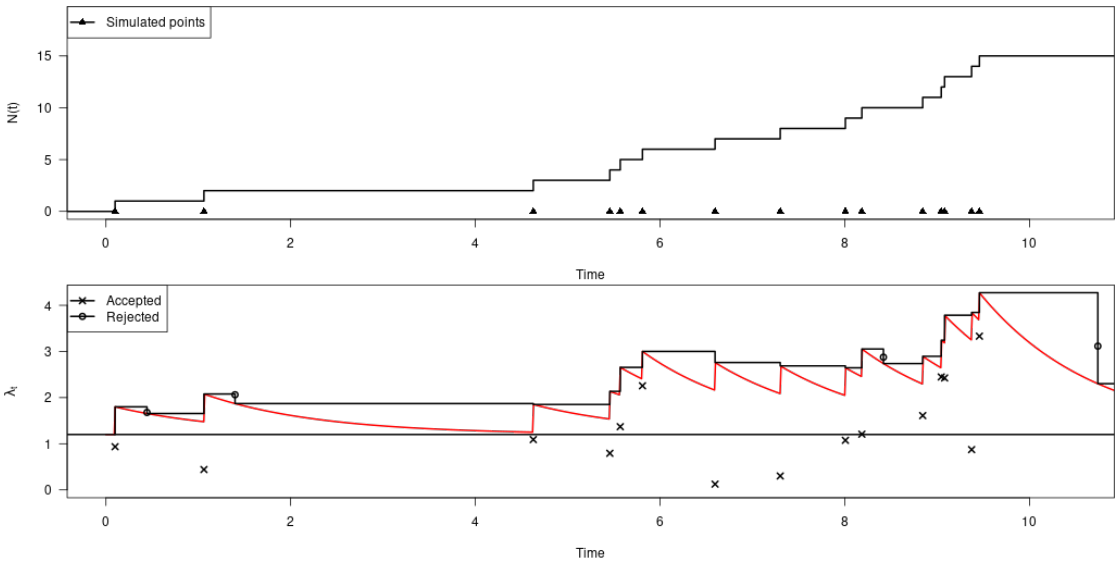


Figure 1.4: **Top**: A realization of a Hawkes process, represented as triangles on the bottom, with intensity function $\lambda(t) = \nu + \int_0^t h(t-s)\,\mathrm{d}N_s$, $\nu = 1.2, h(t) = 0.6\exp -0.8t$. The associated counting process $N(t)$ is represented as well. **Bottom**: The intensity $\lambda(t)$ of the Hawkes process (in red) is drawn below the constructed intensity $\lambda^*(t)$. The uniformly distributed values $u \sim \mathscr{U}[0, 1]$ are displayed, by 'x' when they are accepted by the procedure, and 'o' when they are rejected.

1.3 Biological background

The results presented in this thesis hold for a large variety of stochastic processes, and temporal point processes in particular, and therefore are not bounded to a particular physical case. However, they have been conceived with spiking neural network simulations in mind. We present here the essential knowledge to understand what are the particularities of neural networks that we use in later chapters to optimize the simulation algorithms.

1.3.1 Neurons

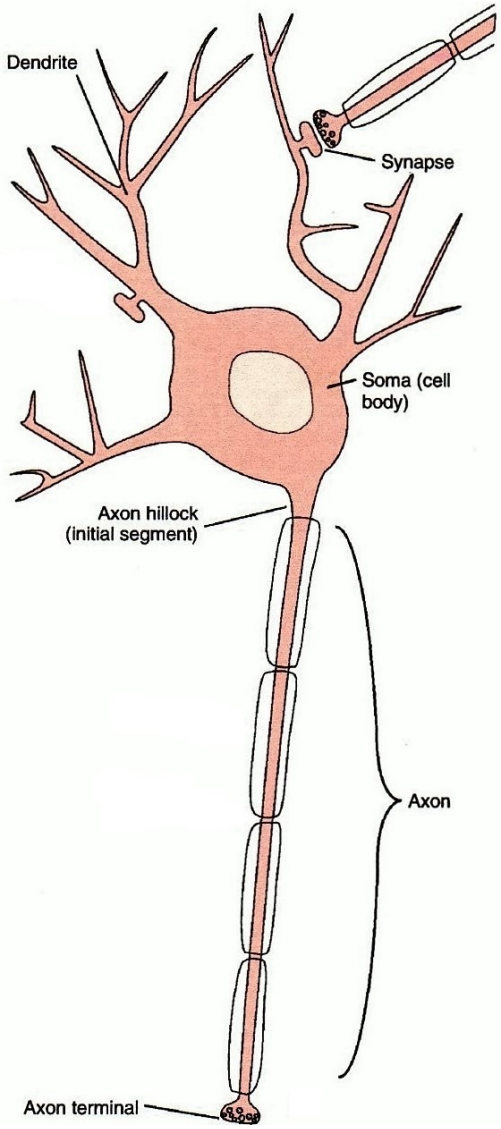
Neurons are a family of cells that are the base constituent of the nervous system in animals [17] (but not the only one, see for instance the glial cells [109]). The purpose of a neuron is to aggregate electrochemical signals from other cells [55, 124], and after potential signal processing [15], to choose whether to emit a signal of its own, called an *action potential*.

There are many types of neurons [9], but in this thesis, and during this brief introduction to the biology of the mammalian central nervous system in particular, we will limit ourselves to the type of neurons that can be found in the mammalian neocortex, namely spiking neurons. Mammalian neurons behave differently from invertebrate neurons [66], as well as from neurons in other parts of a mammalian body, like the analog neurons in the retina and the models we will explore apply primarily to the former.

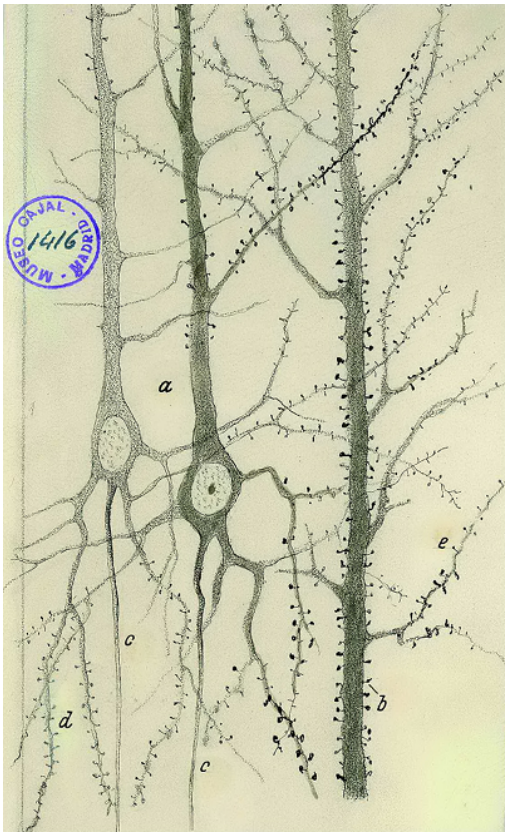
Here we need to introduce some vocabulary relative to neurons and neurons dynamics, so that readers with little biological knowledge about the central nervous system do not get lost. Neurons exchange bursts of electrical current, not alike the one running through the copper wire of electrical devices, but transmembrane electrical potential waves traveling along some parts of the neuron as ions get in and out of their membrane. This traveling electrical burst is called an *action potential*.

The structure of a neuron, illustrated on figure 1.5a, drawn by Cajal on figure 1.5b, reflects the function of this cell: gathering and accumulating action potentials, and releasing action potentials of its own. A typical neuron is composed of three parts: the dendrites, the soma and the axon. The dendrites are a branch-rich structure receiving the action potentials and carrying them to the soma. The latter is the core of the cell, where the electrochemical currents accumulate, more precisely at the axon hillock. Neurons from the cortex let their transmembrane potential accumulate until a certain threshold is reached, which triggers the creation of an action potential that will flow down their axon, where their axon terminal will meet with synapses of other neurons and the signal will be transmitted. See [124, 132] for more details.

The intensity of an action potential does not reflect the amount of action potential energy a neuron has received in order to emit one (at least for cortical neurons). The fact that the intensity



(a) Taken from [100], originally adapted from [124], chapter 3, figure 1, page 43.



(b) Reproduction of a drawing by Cajal of a neuron and its dendrites. From around 1900.

Figure 1.5: The structure of a neuron. Left is a schematic, right is a detailed drawing of a neuron. The soma is easily recognizable, with the nucleus is lighter color than the rest of the cell. The synapses are represented as small protuberances growing from the dendrites and axon. The axon is recognizable as the thick, smooth line descending to the soma, with only few branches departing from it (the cell is flipped vertically compared to the right figure).

of the response of a neuron to a stimulus is not a function of the intensity of the stimulus is called the *all-or-none principle* [124]. A consequence of this principle is that action potentials are very stereotypical objects (a fact at the basis of many models, see [19] for instance), conveying mainly,

if not solely, a timing information. For this reason, a common simplification is to abstract action potentials to a single point, called a spike. A series of spikes during a time window is called a spike train, which is illustrated in figure 1.6. This convenient abstraction motivates the use of point process theory as a model for spike train generation.

1.3.2 Neural networks

If the mechanisms at work when an action potential is generated are well known, it is often difficult to understand how many of them are generated in a live animal during a given unit of time. Superficial measurements such as EEG can only measure the mean electromagnetic field resulting from the total activity of "large" populations of neurons [113]. A non-invasive but still more precise method would be fMRI, but they are not precise enough to allow us to see at the level of individual neurons, not even at the level of a handful of neurons [87]. Besides the non-invasive methods, other means of measuring the neuronal activity is implanting sensors directly in the brain of a live animal, or killing the animal and finely slicing the brain, then to plant sensors in the slices, knowing that neural networks quickly decay after death. Besides the possibility of destroying parts of the environment they are supposed to take measures from, sensors (as well as the non-invasive methods) have the huge drawback of only measuring the activity of close but more importantly *active* neurons. In other words, if during a task most of the neural network stays inactive, or only marginally active, what will be seen will only be the tip of the iceberg, most of the existing neurons/connections remaining invisible to our technological eyes.

This is an important topic because it introduces the issue with neural network studies: they remain unknown objects because of the lack of possible live measurements. If the behavior of individual neurons is at least partially known, direct measurements cannot help determine important parameters of neural networks such as mean spiking rate, connectivity, number of neurons, etc.

Number of neurons

Measuring the number of neurons in the brain of an individual is a tedious task at least, Sisyphean at most. Just in a fruit fly, the number of neurons in the central nervous system is of the order of ten to power five [136]. Count 10^8 for a rat [52], and 10^{11} for a human brain [50]. The famous drawings made by the pioneering neuroscientist Cajal, one of which displayed as figure 1.5b, do not (and cannot) do justice to the density of neurons present in the brain (which makes it all the more impressive that rendering at this scale where already available at that time). Before 2008, the traditional methods involved computing averages on small samples uniformly taken in diverse areas of animal brains, and extrapolating to the whole with a simple cross-multiplication.

These methods gave good order of magnitudes, but failed at delivering more precise results. The new idea successfully experimented in 2008 by Herculano-Houzel [51] was to dissolve parts or even the whole brain to obtain a soup in which the nuclei of neurons would still remain intact. After homogenization, the same technique as before can be used: taking a small sample and counting the neurons inside, except that the sample is now much more representative of the total average. For a human being, it has been estimated that on average about 89 billion neurons were present. In the cortex, a thin layer around the brain of mammals that is widely believed to be an essential part in the production of abstract thoughts [101], Herculano-Houzel *et al.* estimated the number of neurons for humans at around 16 billion [50].

Number of connections

Several types of connections can be considered. Synaptic connections refer to individual connecting synapses, and there are a large number of them between any two neurons (said otherwise, the graph where neurons are vertices and dendrites/axons are edges is not simple). The majority of neuron-to-neuron connections are realized by multiple synapses, at least in the mammalian cortex (see [54] for a list of experimental studies, and chapter 4 for a rough computation of the number of synapse per neuron). Connectomic is the domain trying to map the synaptic connections of a particular brain or a typical member of a specie [32]. To this day, only two species have had their connectome fully mapped at nanoscale: the nematode *C. Elegans* [147] and a tadpole larva of *Ciona intestinalis (L.)* [129]. Both have around two hundred neurons.

Some models, however, consider not the exact synaptic connections but rather the simple existence of even a single synaptic connection between two neurons, effectively aggregating all the individual pathways between two neurons in a single function. The average number of redundant synaptic connections is difficult to compute exactly. In the barrel cortex for instance, it has been estimated that about 10 synaptic connections separate any two cells [37]. This type of experimental data can help create biological constraints to tune better the simulation of biologically constrained neural networks, for instance in [95], where the reconstructed microcircuitry showed an average of 250 neuron-to-neuron connections, which is lower than the 1000 synapses per neuron computed in chapter 4.

The cost of a spike

As stated earlier, it is difficult to measure directly the average activity of the brain, because most of it may be silent during measures and we would not know it. An indirect way to compute this value is to compute the cost of an individual spike, in terms of glucose or ATP molecules input, and measure the amount of glucose the body can provide the brain with, in any given unit of time.

This is a reliable computation, because ATP molecules, once used, are destroyed and cannot be regenerated by the brain, and therefore the body needs to provide constantly the brain with new ATP molecules. The ATP molecules are a necessary input of the spiking mechanism, but also for the basic functioning of a neuron.

Details can be seen in [84]: the cost of each phase of the emission of an action potential gets computed, and the aggregate gives the cost of emitting a single action potential. Assuming all neurons are identical in the brain, or at least the cortex, and knowing the maximum throughput of glucose flowing to the brain each second, the maximum number of spikes per second can be determined for the whole brain and even for parts of it. The figure 1.7 shows the maximum average spike rate sustainable by the human cortex.The energy consumption is expressed as a fraction of the total energy expenditure of an average human brain, and is computed in two different ways, one based on the estimates made by the author of energy availability and costs of neuronal activity (dotted curve), the second (solid curve) being the most conservative estimate to obtain the minimal theoretical energy consumption. Horizontal lines correspond to a percentage of total brain energy consumption. The solid horizontal line corresponds to 44% of the brain energy, which is the accepted value for the cortex [84], and indicates an average of 0.16 spikes/s/neuron. This translates to around 0.30 spikes/s/neuron at the scale of the whole human brain. Higher average spiking rates are therefore impossible to sustain by the brain, energy-wise, according to these calculations.

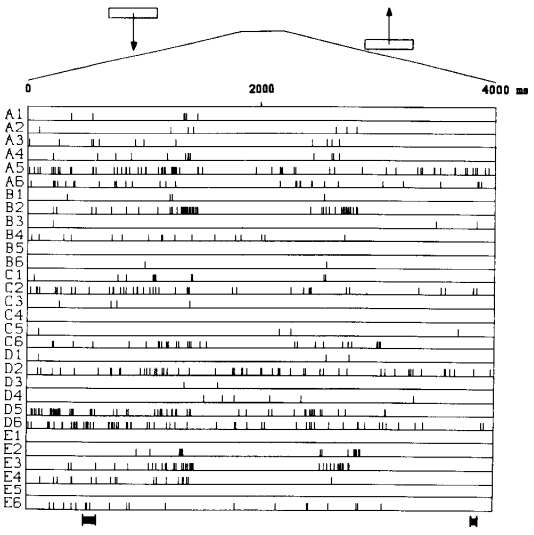


Figure 1.6: An illustration of a spike train. From [74], courtesy of the journal.

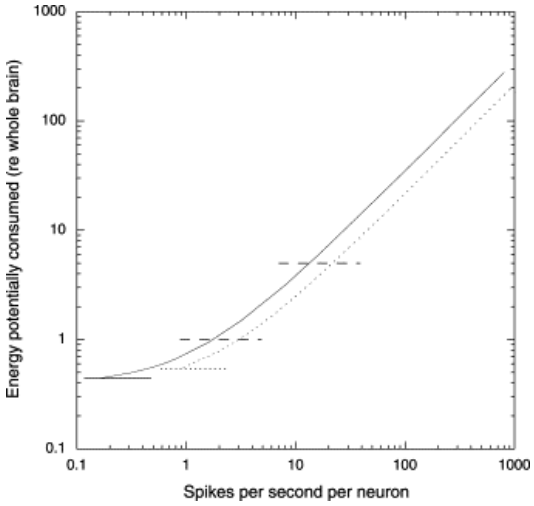


Figure 1.7: Energy potentially consumed by the cortex as a function of the average sustained level of activity in all neurons. From [84], courtesy of the journal.

1.3.3 Biological neuron models

There are a variety of biological or biologically inspired neuron models [19, 20, 88], each serving its own purpose or modeling a particular part of neurons. We are interested in particular in models that use spikes as an abstraction of action potentials. Different models have been proposed, first for single neuron spike train modeling, and later for neural networks (read [38, 39, 63] for possible reviews on the matter).

Leaky-Integrate and Fire (LIF)

Introduced by Lapique in 1907 [1, 19, 82], the Leaky Integrate-and-Fire model, or LIF for short, is a simple set of equations modeling a simplified spiking mechanism. It describes the evolution of the transmembrane potential U_t over time. The idea is simple: the LIF neuron accumulates potential with each received input (bringing it closer to the threshold if the input is excitatory, and farther if the input is inhibitory). However, the storage is not perfect, so in between inputs from other neurons, some of the accumulated potential "leaks" out of the neuron, bringing the transmembrane potential closer to the "resting potential" of the neuron. In the case where the potential reaches at least a certain threshold, a spike would be emitted and the potential of the spiking neuron would be reset to the resting potential.

The equation of the transmembrane potential $u^i(t)$ of neuron i is described by the following set of equations.

$$\frac{\mathrm{d}u}{\mathrm{d}t} = \gamma(u^i(t) - u_R) + RI(t)$$
$$u(t^-) \geq u_T \implies u(t) = u_R$$

γ and R are constant parameters of the model, u_R is called the resting potential of the neuron (this is the value of the potential when no spike has been received by the neuron for a long time, see [65] for biological details), and $I(t)$ is an input current, which can be artificial (a generator) or the spikes of other neurons in the network. When the potential $u(t)$ reaches a certain value u_T, the potential is immediately reset to the resting potential u_R, and the spike is transmitted to other neurons in the network.

The model is one of the foundations of neuroscience [1, 19, 38] and is complex enough to be studied nowadays still. A large variety of variants have since been proposed, such as the noisy LIF [39] or the Generalized Context Model [38]. Other scientists have tried to model more precisely the transmembrane potential dynamic, like the famous Hodgkin and Huxley [55].

Ornstein-Uhlenbeck

The transmembrane dynamic of singular or assemblies of neurons can be described as a random walk, like the Wiener process [81, 125], and are a type of noisy neuron models [39]. The Ornstein-Uhlenbeck model is a modification of the LIF model, with the addition of a Gaussian noise. The model let the potential randomly evolve, though the potential naturally drifts towards an attractor, which can be set below or above the firing threshold. If the attractor value is set above the threshold, the equation models the inclination towards spiking of the neuron (because of unaccounted connections for instance). However, if the attractor value is below the threshold, then only the random variations in the potential or the spikes emitted by its neighbors can make the transmembrane potential cross the firing threshold. The mathematical details of the model can be read in chapter 2.

Hawkes

Point processes have been used for modeling spike trains of single or assemblies of neurons for a long time (see [22, 27, 63, 122, 123] and their references). Introduced by Hawkes [48, 49], the Hawkes point process is a type self-exciting point process (see section 1.2 for details) that has been proposed numerous times for modeling neuronal spiking activity [86].

The Hawkes model of spiking neurons differs from the LIF-inspired models by the fact that no threshold is present that makes neurons spike. This means that as long as the background intensity $\nu > 0$, the neuron modeled as a Hawkes will spontaneously emit spikes. This can either model noise, the basic spiking regime of a neuron during a task, or spontaneous spikes, called "minis" [156]. Also, since the conditional intensity function is analog to an instantaneous firing rate, the intensity of the Hawkes process models the instantaneous spiking rate of neurons, where the equations of the LIF model the subthreshold dynamic of the transmembrane potential, which does not translates directly into a firing rate.

In its linear formulation, the Hawkes process cannot model inhibition, since the conditional intensity function must always be positive. This is an issue since inhibitory neurons constitute a large part of the neurons, around 20% [7]. Non-linear Hawkes process models have been developed to circumvent this limitation. The simplest one is to compose the linear intensity with a ramp function, so that the conditional intensity evaluates to 0 when the linear part is negative.

Artificial Neural Networks

A kind of neural network models not discussed in this thesis are the models designed for learning and classification, also called artificial neural networks. They encompass deep neural networks [2, 62], convolutional neural networks [44], recurrent neural networks [130] and many others.

Maass [89] classifies artificial neural networks in three generations. The first generation is based on the McCulloh-Pitts neurons, also called perceptrons. They can be used in networks, and are universal for digital input and output, meaning that any function mapping a finite set of Boolean variables to another set of Boolean variables can be represented exactly by a network of perceptrons.

Neurons from the second generation use a so-called activation function, mapping a continuous weighted sum of the inputs to a continuous set of outputs. Feedforward neural networks are an example of such a family. The family of second generation of neural networks can represent a higher range of functions that the first generation. Indeed, any neural network from the first family can be represented by a network of the second family, but in addition, second generation neural networks are universal for analog computation. Thus, any continuous function with compact domain and range can be approximated arbitrarily well by a neural network from the second generation. Those neural networks are also suitable for learning using, for instance, a backpropagation algorithm.

Finally, the third generation of neural networks described by Mass [89] are the Spiking Neural Networks [89], which can effectively be any spiking neuron model, as described in [38]. Maas in [89] and the literature cited within, shows that spiking neural networks have a greater computational power than the neural networks from previous generations. They are also universal for digital and analog functions, but in addition smaller neural networks are needed for approximating to the same degree the same functions, at least in the mentioned cases (*ibid*).

Artificial neural network models from the first two generations differ from the models previously explained as the intent behind their design is not the description of physiological properties of neurons, or the statistical reproduction of their spiking activity. However, artificial neural networks are loosely linked to biological neural networks by the fact that both entities can learn to produce a certain response to a given input. Spiking neural networks are an extension of more classical flavors of artificial neural networks that aim at improving the performance of these artificial network. Biological neural networks are indeed energy efficient (see chapter 5), and exhibit remarkable computational performances in general [89, 116].

Though in this thesis we focused our attention on stochastic process models of spiking neurons, the algorithm developed here may also improve the simulation of spiking neural networks in general. The multivariate Hawkes model of a neural network is close to spiking neural network models. They differ from the spiking neuron models presented in [89] as they are stochastic neurons, with spontaneous spiking activity. Note however that spontaneous spikes, called *minis*, with the conjunction of spike-timing dependent plasticity (STDP), to teach a network of spiking neurons a target activity pattern in [58]. Some researchers also try to use the Hawkes neuron model for learning, as in [98].

1.4 Computer science background

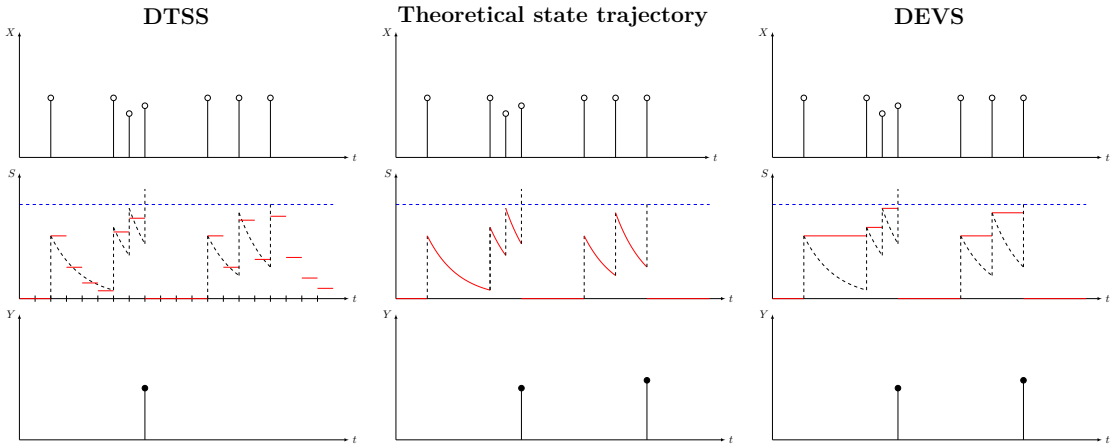


Figure 1.8: A representation of the evolution of the state S of a LIF model (parameters: $\gamma = 1, U_R = 0, U_T = 3$, with input X to the model on top and outputs Y of the model at the bottom. The input section is the same for the three representations of the LIF model. In blue, dashed: the constant threshold U_T. In red, solid: the simulated trajectories. **Middle (LIF)**: the curve is defined by the piecewise differential equation in 1.3.3. The component receives 7 inputs, and emits 2 outputs. Note that the two times the vertical dashed lines, representing the evolution of the state, crossed the threshold U_T, the state $U(t)$ is directly reset at U_R. **Left (DTSS)**: the LIF model defined in 1.3.3 is represented within the DTSS class, and simulated using the Euler method. The time steps are coarse ($\Delta t = 0.5$), to show how the numerical resolution introduces errors. The state $U(t)$ follows loosely the trajectory (dashed black) of the LIF model, represented in the middle column. For the third series of spikes, an output is missed because of the numerical integration, and the trajectory starts diverging from the one defined by the LIF model. **Right (DEVS)**: The LIF modeled and simulated using discrete-events. Note how the state stays constant between two inputs (opposed to the DTSS simulation where the state remains constant only between any two time steps). In this scenario, where the input is scarce, the method is computationally advantageous since there are many fewer events than time steps during the simulation interval. The simulated trajectory is also closer to the true one, and no spike is missed.

Numerical simulations are an important tool for model validation or to better understand how a system works, for instance by looking at discrepancies between simulated results and *in vivo* experimental results. In this section we take a look at the necessary material for understanding the science of simulation programming in the particular case where time is the main variable.

There are three main interconnected entities (sometimes called components) in simulation software: the time management algorithm, the system formalism class and a formulation of the model to simulate fitting within the system formalism. We first present, in this brief introduction, the three main systems formalism classes as defined by [151, 154], then the time management algorithms. The leaky integrate-and-fire (LIF) model is taken as an example, for two reasons. First, efficient simulation algorithms for this particular model have been explored in [16, 104, 154], for

both discrete-time systems specification (DTSS) and discrete-event systems specification (DEVS). Second is that the analytical solution of the differential equation describing the subthreshold dynamic is well known in the computational neuroscience literature [16, 45, 103, 119, 135]. Finally, at the end of this section, we take the time to expose two central concepts in simulation science: pseudo-random number generators and data structures.

1.4.1 System formalisms

The three classes of system formalisms are DESS, DTSS and DEVS. We do not develop the DESS formalism here, since the LIF model cannot be described in terms of DESS because of the discontinuities in the model. However, the differential equations can be numerized and simulated in terms of DTSS or DEVS [43].

DTSS: Discrete-Time System Specification

The DTSS is the family of models defined piecewise with discrete-time increments, such as discretized differential equations and automata. A DTSS model is defined as the set

$$\text{DTSS} = \{X, Y, \mathscr{S}, \delta, \lambda\}$$

where

- X is the set of input to the model
- Y is the set of output from the model
- $\mathscr{S}$ is the set of possible states of the model
- $\delta : \mathscr{S} \times X \to \mathscr{S}$ is the state transition function
- $\lambda : \mathscr{S} \to Y$ is the output function

For instance, a discretized version of the LIF model can use the Euler method with fixed time increments to characterize the state of the model. The sets of input and output remain composed of spikes, but the input and output spiking times t now are multiple of the same time steps $t = k \cdot \Delta t, k \in \mathbb{N}$. Note that this is a source of divergence between the trajectory of a DTSS version of a model and the trajectory of the true model. This divergence, and how to reduce it, is also an active area of research: see for instance [45, 103, 119, 135]. An example is given on figure 1.8, using a very coarse time discretization to exhibit the possible divergence.

DEVS: Discrete-Event System Specification

DEVS is a modeling and simulation framework. Despite using discrete-events, DEVS can be used for modeling both discrete-time *and* discrete-event systems. It was introduced for the first time by Zeigler in 1984 in the first edition of *Theory of Modeling and Simulation* [152], though we refer readers to the more modern version [154].

The formalism creates a clear divide between the model and the simulator. A model in the DEVS formalism can be either *Atomic* or *Coupled*. An atomic model is described by a mathematical structure:

$$AM = \{X, S, Y, \delta_{\text{int}}, \delta_{\text{ext}}, \delta_{\text{conf}}, \lambda, t_a\}$$

where

- X is the set of input values
- $\mathcal{S}$ is the state of states
- Y is the state of output values
- $\delta_{\text{int}} : \mathcal{S} \to \mathcal{S}$ is the transition function for internal events
- $\delta_{\text{ext}} : Q \times X \to \mathcal{S}$ is the transition function for external events, where
  - $Q = \{(s, e) | s \in \mathcal{S}, 0 \leq e \leq t_a(s)\}$
  - $e \in \mathbb{R}^+$ is the time elapsed since the last transition
- $\lambda : \mathcal{S} \to Y$ is the output function
- $t_a : \mathcal{S} \to \mathbb{R}^+$ is the function computing the time until the next internal transition from the current state s
- $\delta_{\text{conf}} : Q \times X \to \mathcal{S}$ is the transition function when there is an internal and an external event happens at the same time

For instance, for a LIF model, the internal event δ_{int} corresponds to the natural return of the potential to the resting value. The external transition function computes what happens when an action potential arrives through the dendrites of the neuron (namely the increase or decrease of the potential). The λ function computes the spike that is emitted when the neuron generates an action potential. The time advance function tells the time until the transmembrane potential reaches the resting potential value. An example of the trajectory of the LIF model defined under DEVS is displayed as the right column of figure 1.8. Note that in this example, we used the analytical solution available for the LIF model, but that other methods exists, such as a quantized version

of the LIF, developped by Kofman *et al* [43]. Indeed, in the general case, the simulation of a continuous system can be achieved by using the quantization of the state described in [72].

Coupling model

Under suitable conditions, any models from one of the three families DTSS, DESS and DEVS can be coupled together, under a structure called a coupling model [154]. This coupling model simply describes the set of atomic models and how they are connected. The DEVS formalism additionally allows the use of input and output ports on the models, so that two models can share multiple connections. Again, examples can be found in [154].

Differences between the three classes

DTSS and DEVS are two families that generally contain a representation of the discretization of a DESS model (or a discrete modeling of a DESS model), or a model based on a system of differential equations in general. The DTSS class represents the models where the time is represented by a clock-driven fashion, while the DEVS class includes the models where time evolves by jumps between discrete-events.
The figure1.8 is a graphical display of the differences between a DTSS, and a DEVS representation of the same model, using the LIF model described in section 1.3.3 as an example. The exact integration of the LIF model has been taken in [128]. Though the DTSS and DEVS version of the LIF are both approximations of the trajectory of the state displayed in the middle column of figure 1.8, the trajectory of the DEVS version of the LIF model stays closer to the theoretical one, because there are less state update, thus less introduction of numerical errors. Note also that in this example the spikes are conveniently placed exactly as multiples of the time step on the left column, but that in practice spikes often end up occurring in between time steps. We recall the work of [45, 103, 119, 135] for a mathematical solution involving interpolation of the spiking time. To keep computational cost low, the solution which is often preferred is to simply choose very thin time steps. This is especially a necessity when each synapse creates its own delay, as explained in [70], where model size was crucial since they used a procedural connectivity, alike the one we proposed in chapters 2 and 4, to fit large-scale neural network on a GPU.

The DEVS formalism may seem cumbersome at first sight, and this is one of the critiques that has been raised against it [140]. However, note that the formalism removes the need for interpretation in the model (which is also a critique formulated in [140]). There is no bijection between the models we create as scientists and the physical world. Furthermore, there is no bijection either between models and their implementation in general, since implementation details focus mainly on choosing the right data structure, the right programming language, sometimes the right hardware

architecture. The liberty left to the scientists implementing the models hinders the reproducibility of results, since different implementations may interpret the models differently. This is also a problem for sharing models within scientific communities, and even worse between scientific communities, since different communities often have different practices, computer science literacy, and history of the software in use.

The DEVS formalism can be a way to reduce these frontiers since it sets a formal framework for defining models that removes modeler interpretation. If the functions and sets are well defined, then any implementation of a DEVS simulator will produce the same simulation output as any other DEVS simulator, given the same stream of random numbers.

We understand, however, that the quality of the formalism does not lie in the execution speed of the simulations implementing it, since the formalism forbids taking design a simulator to take advantage of certain properties of any particular model. Prototyping or proof-of-concept, however, can still be designed using the DEVS formalism, which would facilitate the diffusion of new ideas if the DEVS formalism was more broadly adopted. Research teams could then separately take time for implementing faster versions of a new model or algorithm, knowing on top of that they can rely on the first version of the simulator for the model for testing other kinds of implementations. This is at least a work habit we would like to promote here.

1.4.2 Time management algorithms

Simulation time management can be achieved using either a clock-driven (also called a discrete-time) approach or an event-driven (also called discrete-event) one. They differ by the way time is discretized: cut in batches of fixed size for the discrete-clock scheme, and reduced to a sequence of discrete events for the event-driven strategy. A thorough discussion and historical review on the matter can be found in [154].

Discrete-time simulations

The easiest and most common approach is to discretize time, using either a fixed or variable time step. Starting from an initial condition defined by the modeler, the simulator engine updates the time variable and computes the state of the system at this value of time, then repeats until an ending condition is met. The algorithm 6 illustrates the simulation procedure with discrete-time.

The algorithm relies on two functions, computing the state transition and ending condition, which are applied iteratively starting with a user-specified initial state S_0 at time t_0.. If $\mathscr{S}$ is the set of states, we define the functions transition and continue as:

- transition(S, t):$\mathscr{S} \times \mathbb{R} \to \mathscr{S}$ computes the new state S' after transitioning at time t from state S.

- continue(S,t):$\mathscr{S}\times\mathbb{R}\to\{0,1\}$ tells whether the ending condition is reached if the system is in state S at time t.

Algorithm 6 Stereotypical algorithm for computing discrete-time model trajectories

```
1: t ← t_0
2: S ← S_0
3: while continue(S, t) do
4:     S ← transition(S, t)
5:     t ← t + t_step
6: end while
```

On top of being easy to implement and understand, the algorithm allows the simulation of equations for which an analytical solution or a numerical version is unknown. For discretized differential equations, the algorithm is true at the limit $t_{\text{step}} \to 0$, so the time steps should be chosen small enough to get a good numerical approximation. The algorithm can also easily be computed in parallel if sub-components can be extracted from the system.

The method suffers some drawbacks too, which are important to acknowledge before using it. Due to the fact the time steps should be small, the number of iterations needed to update the system will be large and the rounding errors may become a problem because of the accumulation. Besides the algorithm cannot adapt to the rate of new points it generates, and if the average generation rate is low (see biological background for the average spike rate generation), then most of the computation time will be spent on updating an unchanged system.

To solve some of these issues, another simulation scheme has been invented, which we must know detail further.

Discrete-event simulations

A more subtle solution is to try to update the system only when a change in the state of the system happen. Such possible state change in time is called an event. Note that the change of state is instantaneous, even if the model in continuous time describes a continuous evolution of the state between the time of two events. The algorithm is very similar to the one used for discrete-time simulation, but requires a new function: time_advance$(S,t) : \mathscr{S}\times\mathbb{R}\to\mathscr{S}$. Given the state S of a system and the current time t, the function returns the time t of the next change of state in the system. The function transition$(S,t) : \mathscr{S}\times\mathbb{R}\to\mathscr{S}$ must then be able to compute the transition from state S to the new state. The simulation procedure is described in algorithm 7.

Note that in the case of complex systems with multiple interconnected components (coupled components), the time_advance function must be able to choose which component to prioritize if the output of the time_advance function differs from components to components. The usual choice

Algorithm 7 Stereotypical algorithm for computing discrete-event model trajectories

```
1: t ← t_0
2: S ← S_0
3: while continue(S, t) do
4:     t ← time_advance(S, t)
5:     S ← transition(S, t)
6: end while
```

is to (i) compute the time_advance function independently for each component of the system during initialization, (ii) keep the time of next events sorted at all time in order to, (iii) select the smallest time at each iteration of the algorithm.

The clock-driven and event-driven time management algorithms we just described both present the disadvantage to be applicable to only one type of model (derived from the DTSS class for the clock-driven algorithm, derived from the DEVS class for the event-driven one), with no place for interconnecting models. Furthermore, these algorithms make no assumption on the models being simulated. Different wording of the same models may be incompatible with a specific implementation of these time-management algorithms, or lead to discrepancies in the outputs. If the scientific community focuses solely on these algorithms, model transfers become harder and reproducibility of the results can be an issue. A solution which has been formalized by Zeigler and al. [154] is to create a formal frame for system modeling, which would encompass both discrete-time and discrete-event models.

Time management under coupling

The simulator of a coupled model stores at each time the value of the time advance t_a function for each atomic model in the system, and chooses the smallest for the next transition.

1.4.3 Random numbers

Since computers are deterministic machines, can they compute random numbers? The answer to this question is sadly no [78], but that did not stop mathematicians and computer scientists from trying. The concept of pseudo-random number [78] emerged to describe numbers computed by deterministic algorithms but for which statistical analysis would hardly tell the difference between pseudo and perfectly random numbers.

There is a wide diversity of pseudo-random number generators (PRNG) having been developed by the scientific community. Good examples of scientifically suitable random number generators include the Mersenne-twister [97], gfsr4 [155], Xorshift (with caution [110]), WELL [111]. The Mersenne-twister is actually one of the most widely used generators in scientific computing [79],

being the default choice in programming languages like R, Python and to some extent C++. It is also the pseudo-random number generator used in the work exposed in this thesis.

The following is mostly a summary of what can be found in [76].

Linear Recurrences modulo m

Pseudo-random number generation algorithms work by discrete transition $x_{i-1} \to x_i$ of their internal state $x_.$. The internal state is a set of bytes, or words, that is changed at each new random number generation, and from which the random numbers are computed. The most popular choice for the transition function is a linear recurrence of type

$$x_i = (a_1 x_{i-1} + \cdots + a_k x_{i-k}) \bmod m$$

for some positive k, m and coefficients $(a_i)_i \in \{0, \dots, m-1\}, a_k \neq 0$. The equation is often written in matrix form

$$\mathbf{x}_i = \mathbf{A}\mathbf{x}_{\mathbf{i-1}} \bmod m$$

with $\mathbf{x}_i = (x_{i-k+1}, \dots, x_i)^t$.

A period of $m^k - 1$ can be obtained when m is a prime number and the coefficients a_i are appropriately chosen [71, 75]. The random numbers u_i are computed either by using directly the state x_i of the generator (in which case $u_i \in \{0, \dots, m-1\}$), or by choosing $u_i = \frac{x_i}{m}$ for instance. We focus this review on two particular cases. The case where $k = 1$ is called Linear Congruential Generators (LCG), a classical but obsolete type of PRNG. The infamous RANDU is a member of this family of generator for instance.

The case $m = 2$ is a parametrization that is chosen by a large diversity of modern PRNG, including LFSR, polynomial LCG, GFSR, WELL and the Mersenne-twister. The computation of the output is generally modified as follows:

$$\begin{aligned}
\mathbf{x}_i &= \mathbf{A}\mathbf{x}_{i-1} \bmod 2 \\
\mathbf{y}_i &= \mathbf{B}\mathbf{x}_i \bmod 2 \\
u_i &= \sum_{l=1}^{w} y_{i,(l-1)} 2^{-l}
\end{aligned}$$

where $\mathbf{A}$ and $\mathbf{B}$ are respectively $k \times k$ and $w \times k$ binary matrices, and u_i the i-th random number generated.

Importance of a good PRNG

IBM released in 1968 the now infamous RANDU PRNG, which was widely adopted and used during a decade, because of the lack of instruction of the computer science community about PRNG. We refer curious readers to [94] for an in-depth work of history and analysis about the use of bad PRNG and their consequences. RANDU is now a famously bad generator, which fails most statistical tests that have been developed since then. RANDU is a linear congruential PRNG, which has parameters $a = 65539, c = 0, m = 2^{31}$. The parameters have been explicitly chosen for fast computation of the transition function: $a = 2^{16} + 2^1 + 2^0$, so multiplication by a can be executed with simple bit shifting (16 times and 1 time) and some additions, while the modulo operation can also be executed with bit shifting. Figure 1.9 shows the correlation between any 3 consecutive numbers generated by RANDU. More precisely, if (u_{i-2}, u_{i-1}, u_i) are 3 consecutive numbers generated by RANDU, it is easy to demonstrate that $u_i = [6u_{i-1} - 9u_{i-2}] \mathrm{mod} 2^{31}$. This correlation obviously makes RANDU a very bad random number generator. Note that this is in fact a property of the class of linear congruential generators as demonstrated by Marsaglia in [96].

If a high number of choices exist, some have been tested and used for many years by the scientific community, and therefore should be preferred over newer, less secure ones. In order to assert the quality of a PRNG, some tests have been developed over the years. The NIST gives a battery of tests for testing such software. Easier to use, and maybe of better quality (see chapter 11.8 and 11.9 of [64], also for the NIST suite), the Dieharder test series is a recognized battery of tests composed of ideas from both the scientific community and users. Finally, the "Crush" series ("Small Crush", "Crush" and "Big Crush", see [77]) are among the more thorough tests, though not as up to date as the Dieharder suite since it is older. Note that most PRNG cannot pass all the tests [76], but passing at least as many tests as highly recognized PRNGs such as the Mersenne-twister is the minimum required for a new PRNG to get approved.

A story of seeds

Once a PRNG has been chosen, another important question is seeding. Just like for growing crops, choosing good seeds is an important step to get healthy crops. Indeed, PRNG algorithms are initialized by a seed, a number or a set of numbers provided by the user, sometimes unknowingly. The algorithm then modifies this initial number in order to obtain a new, different number, then repeats the operation on the new number. This series of numbers are called the state of the PRNG, and can be different from what the PRNG will actually output in the end (sometimes the output of the PRNG is the state itself, like for simple linear congruential generators [35, 76]).

Because the streams of random numbers generated depends exclusively on the initial state of the PRNG, it is often important to start the PRNG with a state imbued with enough entropy. For

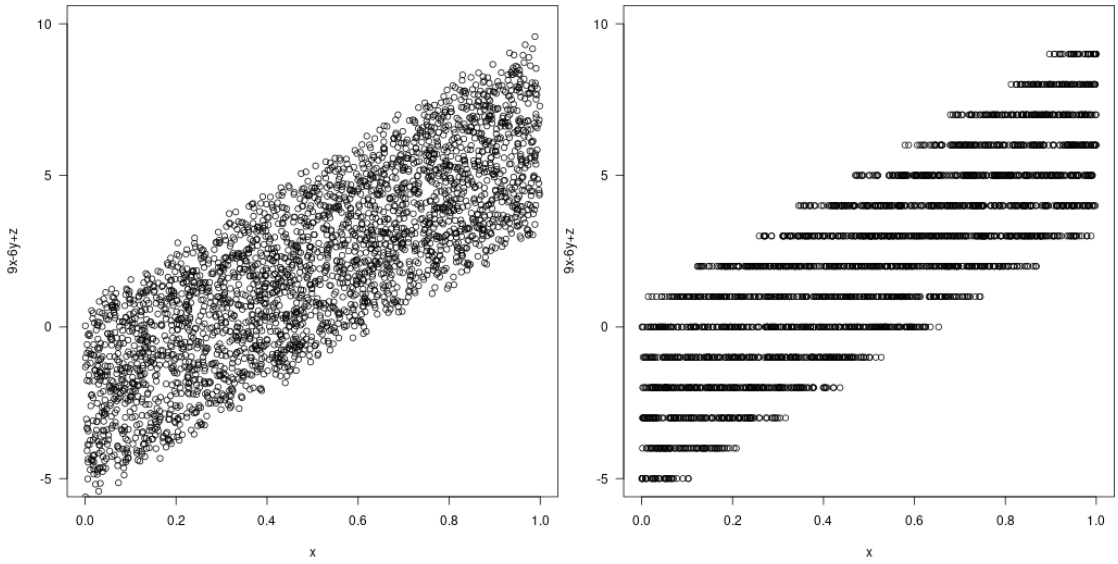


Figure 1.9: Any triplet (x, y, z) of numbers consecutively drawn from the RANDU generator is highly correlated. The combination $9x - 6y + z$ is displayed against the first value x of any randomly generated triplet.**Left**: what the result should look like when the numbers are generated using a good PRNG, here a Mersenne-twister. **Right**: the same function, but now the triplets are generated by successive calls to the RANDU PRNG. The outputs all fall within a small set of hyperplanes.

instance, the initial version of the Mersenne-twister was known to have difficulties recovering from a bad seeding, meaning a seed with a ratio between 0's and 1's far from 0.5 [111].

Entropy-based RNGs and true RNGs

Besides the pseudo-RNGs, another class of RNG exists based on entropy. Collecting data randomly produced by the environment, they are able to generate "true" random numbers [42]. In Linux-based operating systems, the function urandom is one of the interfaces to this random number generator. Their downside lies in their speed: since they collect events from real world devices (mainly the hard drive, but also the keyboard, mouse or thermal noise in the processor chip [42]) in order to generate randomness, these kinds of algorithms are not able to generate too many random numbers at a time (see [42]). This limit is in part due to the fact that these generators use an entropy storage system, which is then used to generate random numbers. When the stock is depleted, it must be refilled before the generator can generate new random numbers. They are therefore not used in Monte Carlo scientific computing. Another reason is reproducibility: since their output is unpredictable, and changes every call, a simulation using such a generator would not be reproducible, unless the stream of numbers were stored along the simulation outputs.

Random number generation for parallel streams

More and more applications are designed with parallel computing in mind. Parallelism is now embedded directly in processors, even single core, through SIMD, while computers have gained more core in the last years and supercomputers now deal with architectures of hundred of thousands of cores. Furthermore, the rise of GPU computing has incentivized even more modelers to embed parallelism in their models and simulation software, especially for Monte Carlo simulations. Not all PRNGs are suitable for parallel simulations. The Mersenne-twister can be adapted for parallel computing, and even GPU computing [139].

1.4.4 Data Structures

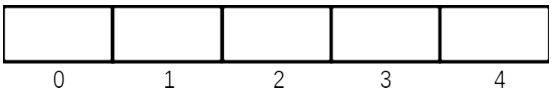
Data structures is one of the most important topics in computer science: computing is processing data, so the study of how to store these data is of prime importance. Or, as the Turing Award laureate Niklaus Wirth titled his now famous entry level book on computer science: Algorithm + data structure = program [148]. We expose here some of the commonly used data structures. We use the notation of the NIST dictionary [11] of data structures, and encourage the reader to read this resource for more information on the matter.

The most basic data structure is the array (illustrated figure 1.10a), which is simply a contiguous area in memory. Elements in an array are identified with their relative index in the array, starting either from 0 or 1 (depending on the language). Arrays are static data structure: once created the size of an array cannot be changed. Elements can be read and modified, given their index in the array. Arrays are often used to implement plain matrices.

A specialization of the static array is the dynamic array, which allows the insertion of new elements at the end. To add an element in the middle of a dynamic array requires shifting the rest of the array to the right.

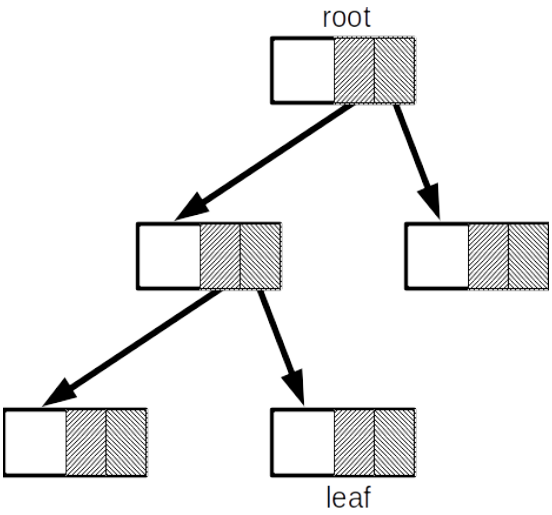
The linked list is a specialized list (illustrated figure 1.10b), a data structure with a head and a tail, and which supports insertion at both ends. It can be defined recursively: a linked list is a data structure which is either 1) empty, or 2) a head node with a reference to a linked list. The linked list additionally supports insertion in the middle without modifying the rest of the data structure. This is possible because each element in the linked list possesses a link to the next element. Each element has its own node which can be situated anywhere in memory. Random access is not possible without reading sequentially the elements from the head or the tail of the list. Linked lists are often used to implement sparse matrices.

The tree (illustrated figure 1.10c), particularly in its balanced specialization, is a middle ground between the sorted dynamic array and the linked list. A tree is usually defined recursively. We define here the binary tree, which is either 1) empty, or 2) a root node linked to a left binary tree



(a) Illustration of an array (or a dynamic array) containing 5 elements. Each element is stored in a square, all the squares being contiguous in memory. Elements are identified by their relative index, here starting at 0 at the left of the array.

(b) Illustration of a linked list with 3 elements. The head and the tail are specified. The white square contains the data, the black square contains the reference to the next node of the linked list.



(c) Illustration of a binary tree with 5 elements. The root and a leaf are identified. Each node is composed of a data segment (white square), and two references, one for the left tree and the other for the right tree. The tree here is balanced: the height of the tree is $3 = \lfloor \log_2 5 \rfloor$.

Figure 1.10: Graphical illustration of some classical and basic data structures.

and a right binary tree. The root node of a tree is the entry point of that tree. It is the only node that has no parents. Trees for which their left and right tree are empty are called leaf. Balanced trees use specific algorithms at insertion and deletion so that no leaf is farther away from the root of the tree than any other leaf.

Binary tree data structures are often designed to keep the elements stored sorted, which is why they are often used in event-driven simulation (see chapter 3).

Algorithms complexity

In computer science, the complexity of an algorithm is a function indicating the amount of resources needed by an algorithm, depending on the data given as input. Complexities are often denoted using big-O notations, with the same intent as how mathematicians use the notation: the resources taken by the algorithm are expected to be of the order of the announced complexity value.

The complexity that is the most often computed is the time complexity of an algorithm, in other words, how long it takes for an algorithm to finish, depending on the input data. The computation of the complexity of an algorithm often requires mathematical insight of the algorithm, see for instance chapter 3.

For the data structure presented in section 1.4.4, the time complexity depends on the number of elements n already present in the structure. Thus, the cost of reading or modifying an element e at index i in an array a is $\mathcal{O}(a.\text{set}(i,e)) = \mathcal{O}(a.\text{get}(i)) = \mathcal{O}(1)$. For the linked list l however, the cost depends on the index of the element: $\mathcal{O}(l.\text{set}(i,e)) = \mathcal{O}(l.\text{get}(i)) = \mathcal{O}(i)$.

The cost of adding an element e at index i of an array of size n depends on its relative position to the end: $\mathcal{O}(a.\text{add\_at}(i,e)) = \mathcal{O}(n-i)$ (without accounting for memory management costs). Indeed, using example of figure 1.10a, adding a new element between position 1 and 2 necessitates replacing the element of index 2 with the new one, then replacing the element in position 3 by the element which was before in position 2, and continue until all elements of index larger than 2 have been shifted to the right ones. Adding an element at the head of a linked list, however, has constant cost $\mathcal{O}(1)$.

Finally, the complexity of an operation looking for an element e or modifying a balanced binary tree t is proportional to the logarithm base 2 of the number of elements in the tree: $\mathcal{O}(t.\text{add}(e)) = \mathcal{O}(t.\text{get}(e)) = \mathcal{O}(\log_2 n)$.

1.5 Software

The algorithms proposed in this thesis have also been implemented, validated experimentally and are available as open source software. The links to get the source code as well as the documentation are given in this section. We also provide some implementation details and a summary of what the software achieve.

GODDESS: Graph-based randOm Discrete-Event Simulator

The GODDESS is an implementation of the DEVS formalism. It is a modified version of the architecture standard proposed by Hu [57]. The goal of the simulator is to provide an efficient and generic DEVS simulator using a simplified architecture. The software uses for instance the base class Object of Java as the base class for all the DEVS objects, in order to remove the need for an artificial base class as proposed in [57].

Some applications have already been implemented in the simulator. The main application is a neural network, using an Input-Brain-Output (IBO) architecture and LIF neuron model. The second main application is a Hawkes point process simulator, which has served as a prototype for testing the applicability of the DEVS formalism for the simulation of point processes models. Another application include a rudimentary real-time DEVS (RT-DEVS) simulator.

URL: https://redmine2.i3s.unice.fr/projects/goddess

Language: Java (simulation) and R (statictical analysis)

Versioning system: git

Ornstein-Uhlenbeck simulator

The Ornstein-Uhlenbeck simulator was prototyped in Java during the 2017 edition of the CEMRACS, then simplified and reimplemented in C. The goal behind this Ornstein-Uhlenbeck simulator is the study of the explosivity of model under certain sets of parameters, for very large networks. A parallel version of the simulator is also available. It uses OpenMP for the parallelization of the biggest tasks, such as the network generation and the main loop. It is also for this simulator that we proposed and successfully experimented a procedural connectivity protocol for the storage of the interaction matrices.

This simulator has been used to obtain in part the results of the article presented in chapter 2. The simulator opens the way for using discrete-event techniques for the improving the simulation algorithm of stochastic processes.

URL: https://github.com/mascartcyrille/cemracs
Language: Java and C (simulation)
Versioning system: git

SPIKES: Sparse graPh multIvaried hawKes point procEss Simulation

SPIKES is the descendant of the Hawkes simulator implemented in GODDESS, but uses C++ as main language instead of Java. The goal was to obtain a more efficient simulator, in terms of both speed and memory, than the one available in GODDESS. Since the first objective was the comparison between our newly proposed algorithm and the classical algorithm by Ogata (see section 1.2.8), we also wanted to remove the parts pertaining to the DEVS formalism and that were unnecessary for the simulation of a Hawkes process. The main ideas (separation of simulator and model, discrete-event based simulator, internal and external events) are still present, though some names have been changed.

A set of tools for statistical analysis of point processes, based on the UnitEvents R package, is also available in the library. It is a series of goodness-of-fit tests for validating that the simulator accurately simulates a Hawkes process. This can be used for test-driven implementation, or when changes are made to the simulator for instance.

URL: https://gitlab.inria.fr/neuromod/nm-spikes
Language: C++ (simulation), R (statistical analysis)

Versioning system: git

1.6 Organization

This introduction constitutes the chapter 1 of this document, while the rest of the thesis is composed of three chapters, along with the conclusion. The three chapters are the transcript of articles that have been submitted to peer-reviewed journal. Only the first one has been fully accepted yet, and is published in ESAIM: Proceedings and Surveys. This is chapter 2 of the manuscript, which deals with the efficient simulation of large-scale stochastic process. The chapter 3 has been accepted in TOMACS, at the condition of minor modifications. Chapter 4 is still under the peer-review process. These two chapters deal with the efficient simulation of point process, and Hawkes point process in particular. At last, the conclusion synthesizes the whole thesis work and broaches on open questions and future works.

2 Article 1: Network of interacting neurons with random synaptic weights

Written by P. Grazieschi, M. Leocata, C. Mascart, J. Chevallier, F. Delarue, E. Tanré

In this article we explored the dynamic of a model of a network of N spiking neurons. The transmembrane potential of individual neurons is modeled using a stochastic version of the Leaky Integrate and Fire, called Ornstein-Uhlenbeck. The interactions are instantaneous in this model, which can create singularities (an infinite amount of spikes can be emitted in a finite time). The goal is to search for initial conditions that can lead to singularities, in particular, how the connectivity and the connections weights condition the singularity.

The article is divided in two parts. First a collection of mathematical results on the behavior of the solution, when the synaptic weights are random and the connection graph is an instance of an Erdös-Renyi graph. The second part focuses on the algorithm that can be used for simulating this model.

My contributions to this article is the creation and implementation of an efficient discrete-event algorithm in the second part. In particular, a new algorithm and a data structure for procedural connectivity are proposed at the end of the article, along with the computation of the theoretical complexity values for memory and time.

Status: Submitted and published in ESAIM

Abstract

Since the pioneering works of Lapicque [16] and of Hodgkin and Huxley [15], several types of models have been addressed to describe the evolution in time of the potential of the membrane of a neuron. In this note, we investigate a connected version of N neurons obeying the leaky integrate and fire model, previously introduced in [1, 2, 3, 6, 7, 14, 17, 19, 21]. As a main feature, neurons interact with one another in a mean field instantaneous way. Due to the instantaneity of the interactions, singularities may emerge in a finite time. For instance, the solution of the corresponding Fokker-Planck equation describing the collective behavior of the potentials of the neurons in the limit $N \to \infty$ may degenerate and cease to exist in any standard sense after a finite time. Here we focus out on a variant of this model when the interactions between the neurons are also subjected to random synaptic weights. As a typical instance, we address the case when the connection graph is the realization of an Erdös-Renyi graph. After a brief introduction of the model, we collect several theoretical results on the behavior of the solution. In a last step, we provide an algorithm for simulating a network of this type with a possibly large value of N.

2.1 Introduction

2.1.1 From a model for one neuron to a model for N neurons

One of the first models for neurons was introduced by Louis Lapicque in 1907, see [16]. It is called Integrate and Fire (IF) model. The membrane potential $(X(t))_t$ of a neuron evolves in time according to the simple electrical equation

$$I(t) = C\frac{dX(t)}{dt},$$

where $(I(t))_t$ is an input current that goes through the membrane. The above equation holds true up until the neuron emits a *spike*. We use the generic notation τ_F for the *spiking time*. From the mathematical point of view, τ_F is regarded as the first time when the membrane potential reaches a certain hard threshold value X_F, the latter being referred to as a *firing value*. Equivalently, the time τ_F is nothing but the first *hitting time* of X_F. As just mentioned, it must be seen as a *firing time* at which the neuron emits a *spike*. After τ_F, the dynamics are continued in the following way: At time τ_F, the potential is reset to a reset value X_R and the dynamics are refreshed from X_R. This model serves as the basis for a large variety of other neural network models developed in order to model more accurately certain behaviors of neurons and neural networks, such as

memory, leaking, etc. For instance, a common feature is to write the *sub-threshold dynamics* of the membrane potential in the form of a stochastic (instead of ordinary) differential equation:

$$dX_t = b(X_t)dt + I_t dt + dW_t,$$

for $t < \tau_F$, where $(I_t)_t$ describes the mean trend of the input current (we here use the probabilistic convention: the time parameter is put in index) and $(\frac{d}{dt}W_t)_t$ is a white noise accounting for fluctuations in the input. A typical choice for the coefficient b is $b(x) = -\lambda(x-a)$, in which case λ reads for some leakage parameter. This model is commonly referred to as a stochastic LIF model.

In this article, we address a network of N connected neurons that evolve according to a variant of the above stochastic LIF model. We describe the discrete system of neurons through the evolution in time of the membrane potentials of each of the neurons: The membrane potential of neuron i (for $i = 1, \cdots, N$) is described by a process $X^i = (X_t^i)_{t\geq 0}$. The *sub-threshold* dynamics of each X^i are of the general form:

$$dX_t^i = b\big(X_t^i\big)dt + dI_t^{i,\,\text{network}} + dW_t^i, \qquad t \geq 0, \quad i = 1, \cdots, N,$$

with initial conditions $X_0^i < X_F$. Here $(X_0^i)_{i=1,\cdots,N}$ is a family of independent and identically distributed random variables and $(W^i)_{i=1,\cdots,N}$ is a family of independent Brownian motions, the initial conditions and the noises being independent. The term $dI_t^{i,\,\text{network}}$ models the current that neuron i receives at time t from the other neurons. As above, the value X_F is a firing threshold, which is assumed to be common to all the neurons: Whenever a neuron reaches the threshold value X_F, its potential is reset to X_R, X_R being the same for all the neurons.

For sure, the term $dI_t^{i,\,\text{network}}$ is of great importance in the description of the model. Inspired by [17, 21], we choose $dI_t^{i,\,\text{network}}$ in the form

$$dI_t^{i,\,\text{network}} = \sum_{j=1}^{N} \sum_{k\geq 1} J_N^{j\to i} d\delta_0\big(t - \tau_k^j\big),$$

where δ_0 is the Dirac delta measure in 0, τ_k^j (for some $j = 1, ..., N$ and $k \in \mathbb{N}$) is the k-th time at which the potential of neuron j reaches the threshold value X_F, and $J_N^{j\to i}$ is a *synaptic weight* that prescribes the influence of neuron j onto neuron i. In case when the synaptic weight is positive, the interaction from j to i is excitatory; otherwise, it is inhibitory. It is worth mentioning that, with the above prescription, $(dI_t^{i,\,\text{network}})_t$ is not exactly a current since the measure $dI_t^{i,\,\text{network}}$ is singular. Differently, the term $dI_t^{i,\,\text{network}}$ account for impulses received by neuron i from the network at time t. These impulses account for the instantaneity of the interactions: each time neuron j spikes, neuron i feels a pulse of intensity $J_N^{j\to i}$.

The main purpose of this note is to address the case when the synaptic weights are of the form

$$J_N^{j\to i} = \frac{\beta}{N}\alpha^{i,j},$$

where β is a scaling parameter that calibrates the global connectivity of the network and $\alpha^{i,j}$ represents some variability in the synaptic weight between neurons i and j. The factor $1/N$ is typical of a *mean field* model. The main feature is that we allow the weights $(\alpha^{i,j})_{i,j=1,\cdots,N}$ to be random, in which case the tuples $(X_0^i)_{i=1,\cdots,N}$, $((W_t^i)_{t\geq 0})_{i=1,\cdots,N}$ and $(\alpha^{i,j})_{i,j=1,\cdots,N}$ are required to be independent.

The idea below is to study the behavior of the system, when N is large, considering different kind of randomness for $\alpha^{i,j}$.

2.1.2 Mean field limit

The time integrated version of the dynamics may be put in the more accessible form:

$$\begin{cases} X_t^i = X_0^i + \int_0^t b\big(X_s^i\big)ds + \sum_{j=1}^N \sum_{k\geq 1} J_N^{j\to i}\mathbb{1}_{\{\tau_k^j\leq t\}} + W_t^i, & \text{if } X_{t-}^i < X_F, \\ X_t^i = X_R, & \text{if } X_{t-}^i = X_F. \end{cases} \tag{2.1}$$

The case $J_N^{j\to i} = \alpha/N$, for a deterministic factor α, has been already addressed in several papers, among which [1, 2, 3, 6, 7, 17, 21]. In all these references, authors have paid much effort to discuss the asymptotic regime $N \to \infty$.

Because of the normalization factor α/N in the interaction term and by independence of the various sources of noise, we may indeed expect from standard results on large particle system with mean field interaction, see for instance [23], that neurons become independent asymptotically, each of them solving in the limit a nonlinear stochastic differential equation with interaction with the common theoretical distribution of the network. In the limiting network, the membrane potential $(X_t)_t$ of a *representative neuron* should evolve according to the equation:

$$\begin{cases} X_t = X_0 + \int_0^t b\big(X_s\big)ds + \alpha\mathbb{E}\big[\sum_{k\geq 1}\mathbb{1}_{\{\tau_k\leq t\}}\big] + W_t, & \text{if } X_{t-} < X_F, \\ X_t = X_R, & \text{if } X_{t-} = X_F, \end{cases} \tag{2.2}$$

where $W = (W_t)_t$ is the proper noise to the representative neuron and $(\tau_k)_k$ stands for its own *spike train*. Passage from (2.1) to (2.2) is usually referred to as *propagation of chaos* in the literature. While it may be simpler to justify for systems with regular interactions, it turns out to be a much more challenging problem in the current framework because of the instantaneity of the interactions between the neurons.

In fact, even equation (2.2) itself –which should be called a McKean Vlasov equation– is a difficult object to study, especially in the excitatory regime $\alpha > 0$. In order to address its well-posedness, two approaches are conceivable, a PDE one and a probabilistic one. The PDE approach is based upon the description, through the so-called Fokker-Planck equation, of the evolution of the marginal law of the potential. We refer to [1, 2, 3]. The probabilistic route is to analyze directly equation (2.2), see [6, 7] and the more recent contributions [14, 19]. In fact, both approaches face the same difficulty: Whenever the interaction parameter α is too large, the system may **blow up** in some sense. Heuristically, a blow-up occurs when a large proportion of neurons in the finite network spike at the same time; in the limit regime, a blow up occurs when the time derivative of the mean field term in (2.2) is infinite. It is proven in [1, 6] that a blow up appears if the mass of the initial distribution is too concentrated near the firing threshold X_F and, conversely, that no blow up appears if the initial mass is sufficiently far away from X_F. While global existence of solutions with a blow up is addressed in [7] local uniqueness results are shown in [6, 14, 19]: In these latter references, successive improvements are obtained on the length of the interval on which uniqueness is known to hold.

Blow up not only matters for the limiting mean field equation. Going back to the particle system, we understand that the difference between the two values X_F and X_R is certainly important because it must influence, among others, the typical delay between two consecutive spikes. In particular, when α is large in comparison with $X_F - X_R$, a neuron may spike at a high frequency because of the pulses received from the others. This stylized fact is at the core of our analysis. Still, the reader must be aware that, in biological models, an additional refractory period is usually added, during which a neuron is somehow locked after a spike. Here we shall not include such a refractory period; for sure, it would ask for further investigations.

Whilst the case $J_N^{j\to i} = \alpha/N$ has received much attention in the literature, the case of random synaptic weights has received limited attention. This is our precise objective to understand how randomness in the synaptic weights may impact the passage from (2.1) to (2.2) and how it may impact the emergence of a blow up in the system. In this regard, it is worth noticing that another form of inhomogeneous connection is addressed in [18]: Therein, a finite network of distinct infinite populations is studied; inhomogeneous weights are used to describe the connections between populations.

2.1.3 Model with random synaptic weights

Throughout the text, we will take $X_F = 1$ and $X_R = 0$. This permits to rewrite equation (2.1) for the network of $N \in \mathbb{N}$ neurons in the following more concise form:

$$X_t^i = X_0^i + \int_0^t b(X_s^i)ds + W_t^i + \sum_{j=1}^{N} J_N^{j\to i} M_t^j - M_t^i, \tag{2.3}$$

where the process M_t^i counts the number of times that the potential of neuron i has reached the threshold value $X_F = 1$; formally, we may write

$$M_t^i = \int_0^t \sum_{k\in\mathbb{N}} d\delta_0(s - \tau_k^i) = \sum_{k\in\mathbb{N}} \mathbb{1}_{[0,t]}(\tau_k^i),$$

with τ_k^i standing for the random time at which the process X^i reaches the threshold for the k-th time.

As already announced, we will discuss the behavior of the system for different types of variables $\{J_N^{j\to i}\}_{i,j=1,\cdot,N}$. Particularly, we will address the case $J_N^{j\to i} = \beta\,\alpha^{i,j}/N$, with $\beta > 0$ constant and:

1. $\alpha^{i,j} \sim \mathscr{B}(p)$ i.i.d.;
2. $\alpha^{i,j}$ dependent with $\alpha^{i,j} = \alpha^i\alpha^j$ and $\alpha^i \sim \mathscr{B}(p)$ i.i.d.;
3. $\alpha^{i,j}$ dependent with $\alpha^{i,j} = p\alpha^i$ and $\alpha^i \sim \mathscr{B}(p)$ i.i.d.;
4. $\alpha^{i,j} \sim \mathscr{B}(p_N)$ i.i.d. with $\lim_{N\to\infty} p_N = 0$.

The note is organized as follows. We provide some theoretical results in Section 2.2. Section 2.3 is dedicated to the presentation of an algorithm for simulating (a variant of) the particle system with a large value of N. Throughout the text, all the Brownian motions are normalized, meaning that their volatility is 1.

2.2 Mathematical inquiries

This section is dedicated to a theoretical analysis of (2.3), choosing $X_F = 1$ as firing threshold and $X_R = 0$ as firing potential. Throughout the section, we discuss (sometimes in a pretty informal way) the form of the limiting mean field equation together with the existence of a blow-up, as defined in the previous section.

2.2.1 Independent Random weights: alphaîsim mathcalB(p)

Let us consider the case where the connections are i.i.d. Bernoulli variables, with a parameter p that is independent of N, namely $\alpha^{i,j} \sim \mathscr{B}(p)$ (this is case (1) in Subsection 2.1.3). From a modeling point of view, this amounts to say that the neurons are connected through the realization of an Erdös-Renyi graph. The first purpose is to conjecture what is the limit equation, or equivalently the analogue of (2.2), for the network.

Observe first that, whenever $p = 1$, the synaptic weight takes the form $J^{j\to i} = \beta/N$, which exactly fits the case addressed in (2.2). By some heuristic calculation, we expect that, in the more general case $p \in (0, 1]$, the limiting mean field term manifests in the form:

$$p \cdot \beta \cdot \mathbb{E}\big[M_t\big].$$

Intuitively, the reason is that, as $N \to \infty$, not only the neurons should become asymptotically independent between them, but also they should become independent of the synaptic weights. Indeed a given neuron i feels less and less the values of each $\alpha^{i,j}$ as N grows up; this means that, applying a law of large numbers, the interaction term in (2.3) should get closer and closer to

$$\frac{\beta}{N}\mathbb{E}\left[\sum_{j=1}^{N}\alpha^{i,j}M_t^j\right] = \frac{\beta}{N}\mathbb{E}\left[\int_0^t\sum_{j=1}^{N}\sum_k \alpha^{i,j}d\delta(s-\tau_k^j)\right] = \frac{\beta}{N}\mathbb{E}\left[\sum_{j=1}^{N}\sum_k \alpha^{i,j}\mathbb{1}_{\{\tau_k^j\le t\}}\right] \approx p\cdot\beta\cdot\mathbb{E}[M_t].$$

So, as a limit equation for the network, we expect the following (at least in the *sub-threshold regime*):

$$X_t = X_0 + \int_0^t b(X_s)ds + \beta\cdot p\cdot\mathbb{E}[M_t] + W_t - M_t, \tag{2.4}$$

with $M_t = \sum_{k\in\mathbb{N}}\mathbb{1}_{[0,t]}(\tau_k)$. In other words, the Bernoulli parameter should scale as a linear factor in the intensity of the interaction. In particular, if our derivation is correct, the limiting equation fits the framework considered in [1, 2, 3, 6, 7]. We will come back to this point next.

For the time being, we have in mind to justify more rigorously the conjectured equation. To do so, we propose the following argument based on a simple model for propagation of chaos.

2.2.2 On the Derivation of the limit equation in a toy model

To get an idea of the behavior of particles in our setting, we consider the simpler model in which the processes $(X^i)_i$ satisfy the following system of equations:

$$dX_t^i = \frac{1}{N}\sum_{j=1}^{N}\alpha^{i,j}X_t^j\,dt + dW_t^i, \quad i = 1,\cdots,N, \tag{2.5}$$

where $(\alpha^{i,j})_{i,j=1,\dots,N}$ is a family of i.i.d. bounded random variables and $(W^i)_i$ is a family of independent standard Brownian motions. The families $(\alpha^{i,j})_{i,j}$ and $(W^i)_i$ are assumed to be independent. For simplicity, we do not address the influence of the threshold and do as if X_F was equal to $+\infty$. Also, we take the same deterministic initial conditions $X_0^i = x_0$ for all the particles.

We use the technique of coupling introduced in [23]. That is, we want to show convergence of (2.5) to

$$d\bar{X}_t = \mathbb{E}[\alpha]\mathbb{E}[\bar{X}_t]dt + dW_t, \quad \bar{X}_0 = x_0, \tag{2.6}$$

where α is another random variable with the same distribution as each $\alpha^{i,j}$ and W is a standard Brownian. Although it is not needed to state the equation, we assume for convenience that α and W are independent and that (α, W) is independent of the family $((\alpha_{i,j})_{i,j}, (W_i))$. To prove the convergence stated above, let us also introduce, for each $i \in \{1, \cdots, N\}$, the SDE:

$$d\bar{X}_t^i = \mathbb{E}[\alpha]\mathbb{E}[\bar{X}_t^i]dt + dW_t^i, \quad \bar{X}_0^i = x_0, \tag{2.7}$$

and let us follow the strategy consisting in comparing (2.6) with (2.7) and (2.7) with (2.5).

As for (2.6) with (2.7), we notice that $\bar{X}$ and $\bar{X}^i$ have the same law. We therefore just need to consider X^i and $\bar{X}^i$. For the latter point, we observe that the processes $(\bar{X}, (\bar{X}^i)_i)$ are independent of the family $(\alpha^{i,j})_{i,j}$. Using the independence of α and $\bar{X}^i$, we also know that

$$\begin{aligned} d\big(X_t^i - \bar{X}_t^i\big) &= \left(\frac{1}{N}\sum_{j=1}^N \alpha^{i,j}X_t^j - \mathbb{E}[\alpha\bar{X}_t^i]\right)dt \\ &= \left[\left(\frac{1}{N}\sum_{j=1}^N \alpha^{i,j}X_t^j - \frac{1}{N}\sum_{j=1}^N \alpha^{i,j}\bar{X}_t^j\right) + \left(\frac{1}{N}\sum_{j=1}^N \alpha^{i,j}\bar{X}_t^j - \mathbb{E}[\alpha\bar{X}_t^i]\right)\right]dt. \end{aligned}$$

Since $\alpha^{i,j}$ is bounded by a constant C, we may write

$$\big|X_t^i - \bar{X}_t^i\big| \le \int_0^t \frac{C}{N}\sum_{j=1}^N \big|X_s^j - \bar{X}_s^j\big|ds + \int_0^t \left|\frac{1}{N}\sum_{j=1}^N \alpha^{i,j}\bar{X}_s^j - \mathbb{E}[\alpha\bar{X}_s^i]\right|ds.$$

By summing over i and dividing by N, also defining δ_t to be the function $\frac{1}{N}\sum_{i=1}^N \big|X_t^i - \bar{X}_t^i\big|$, we get

$$\delta_t \le C\int_0^t \delta_s ds + \frac{1}{N^2}\int_0^t \sum_{i=1}^N \left|\sum_{j=1}^N \Big(\alpha^{i,j}\bar{X}_s^j - \mathbb{E}[\alpha\bar{X}_s^i]\Big)\right| ds.$$

By using Gronwall's lemma, we then deduce that

$$\delta_t \leq \frac{\exp(Ct)}{N^2} \int_0^t \sum_{i=1}^N \left| \sum_{j=1}^N \left(\alpha^{i,j} \bar{X}_s^j - \mathbb{E}\left[\alpha \bar{X}_s^i\right] \right) \right| ds.$$

Taking the expectation, we have the following:

$$\mathbb{E}[\delta_t] \leq \frac{\exp(Ct)}{N} \int_0^t \sum_{i=1}^N \left(\frac{1}{N} \mathbb{E} \left| \sum_{j=1}^N \left(\alpha^{i,j} \bar{X}_s^j - \mathbb{E}\left[\alpha \bar{X}_s^i\right] \right) \right| \right) ds.$$

Moreover, it is now useful to investigate the second moment of $\left(\alpha^{i,j} \bar{X}_s^j - \mathbb{E}\left[\alpha \bar{X}_s^i\right] \right)$, which is a finite value C_2^2. Hence, by independence of $(\alpha^{i,j})_{i,j}$ and $(\bar{X}^j)_j$, we get:

$$\mathbb{E}[\delta_t] \leq \frac{\exp(Ct)}{N} \int_0^t \sum_{i=1}^N \frac{C_2}{N^{1/2}} ds \leq \exp(Ct) \frac{C_2 t}{N^{1/2}}.$$

Since it holds that

$$\delta_t \geq W_1 \left(\frac{1}{N} \sum_{i=1}^N \delta_{X_t^i}, \frac{1}{N} \sum_{i=1}^N \delta_{\bar{X}_t^i} \right),$$

where $W_1(\cdot, \cdot)$ is the 1-Wasserstein distance defined as $W_1(\mu, \nu) = \inf_\pi \int_{\mathbb{R}^2} |x - y| d\pi(x, y)$, the infimum being taken over all the probability measures π on $\mathbb{R}^2$ that have μ and ν as marginal distributions, we have also proved that, for any $T > 0$,

$$\lim_{N \to \infty} \sup_{0 \leq t \leq T} \mathbb{E} \left[W_1 \left(\frac{1}{N} \sum_{i=1}^N \delta_{X_t^i}, \frac{1}{N} \sum_{i=1}^N \delta_{\bar{X}_t^i} \right) \right] = 0.$$

By the law of large numbers, this implies that

$$\lim_{N \to \infty} \sup_{0 \leq t \leq T} \mathbb{E} \left[W_1 \left(\frac{1}{N} \sum_{i=1}^N \delta_{X_t^i}, \mathscr{L}(\bar{X}_t) \right) \right] = 0,$$

which is nothing but propagation of chaos.

2.2.3 Blow-up Argument

We now address the blow-up phenomenon for (2.4). Since (2.4) is similar, up to the scaling factor p, to the equation investigated in [1, 6], we expect the same picture: If the initial condition is too close to $X_F = 1$, then a blow-up occurs; if it is sufficiently far away, then it cannot occur. Here we show that a blow-up indeed exists if $X_0 = x_0$ for some deterministic x_0 that is close enough to 1.

Our proof is based upon the probabilistic approach, but, in fact, it consists of a reformulation of the arguments used by Cáceres et al. in [1]. Below, we use repeatedly the notation $e(t) := \mathbb{E}[M_t]$.

Theorem 3. *Assume that drift coefficient satisfies, for some $\lambda > 0$,*

$$b(v) \geq -\lambda v, \quad \text{for all } -\infty < v \leq 1.$$

If the (deterministic) initial condition $X(0) = x_0 < 1$ is sufficiently close to the threshold 1, there is no global in time solution of the SDE

$$\begin{aligned} dX_t &= b(X_t)\,dt + \beta\, p\, e'(t)\,dt + dW_t - dM_t, \qquad t \geq 0, \\ \tau_{k+1} &= \inf\{t \geq \tau_k, X_t \geq 1\} \\ M_t &= \sum_{k\geq 1} \mathbb{1}_{[0,t]}(\tau_k) \end{aligned} \tag{2.8}$$

with $e(t) = \mathbb{E}[M_t]$.

Proof. The idea is to retrace the proof in [1]. Assume indeed that we have a global in time solution $(X_t)_{t\geq 0}$ to (2.8) with a differentiable mean counter e. The object of our study is then $F_\mu(t) := \mathbb{E}[\varphi(X_t)]$ with $\varphi(x) = \exp(\mu x)$, where

$$X_t = X_0 + \int_0^t \Big(b(X_s) + \beta\, p\, e'(s)\Big)ds + W_t - M_t.$$

Now we apply Itô's formula to $\varphi(X_t)$:

$$\varphi(X_t) = \varphi(X_0) + \int_0^t \underbrace{\varphi'(X_s)}_{\mu\varphi(X_s)} \Big(b(X_s) + \alpha \cdot pe'(s)\Big)ds + \int_0^t \underbrace{\varphi'(X_s)}_{\mu\varphi(X_s)}\, dW_s + \frac{1}{2}\int_0^t \underbrace{\varphi''(X_s)}_{\mu^2\varphi(X_s)}\, ds + \int_0^t \underbrace{[\varphi(X_{s-} - 1) - \varphi(X_{s-})]}_{\big(\varphi(0)-\varphi(1)\big)}\, dM_s.$$

Taking expectation:

$$\mathbb{E}\big[\varphi(X_t)\big] = \mathbb{E}\big[\varphi(X_0)\big] + \mu\int_0^t \mathbb{E}\Big[\varphi(X_s)\Big(b(X_s) + \beta \cdot pe'(s)\Big)\Big]ds + \frac{\mu^2}{2}\int_0^t \mathbb{E}\big[\varphi(X_s)\big]ds + \Big(\varphi(0) - \varphi(1)\Big)e(t).$$

Recalling that $F_\mu(t) = \mathbb{E}[\varphi(X_t)]$, we can rewrite the above expression as:

$$F_\mu(t) = F_\mu(0) + \mu\int_0^t \mathbb{E}\Big[\varphi(X_s)\Big(b(X_s) + \beta \cdot pe'(s)\Big)\Big]ds + \frac{\mu^2}{2}\int_0^t F_\mu(s)ds + \Big(\varphi(0) - \varphi(1)\Big)e(t).$$

Then, using the hypothesis on b, we get that

$$F_\mu(t) \geq F_\mu(0) + \mu \int_0^t \mathbb{E}\big[\varphi(X_s)(\beta \cdot pe'(s) - \lambda X_s)\big] ds + \frac{\mu^2}{2} \int_0^t F_\mu(s) ds + \Big(\varphi(0) - \varphi(1)\Big) e(t)$$
$$\geq F_\mu(0) + \mu \int_0^t (\beta \cdot pe'(s) - \lambda) F_\mu(s) ds + \frac{\mu^2}{2} \int_0^t F_\mu(s) ds + \Big(\varphi(0) - \varphi(1)\Big) e(t),$$

that is

$$F_\mu(t) \geq F_\mu(0) + \int_0^t \mu\Big(\beta \cdot pe'(s) - \lambda + \frac{\mu}{2}\Big) F_\mu(s) ds + \Big(\varphi(0) - \varphi(1)\Big) e(t). \tag{2.9}$$

Define $\tilde{\lambda}$ as

$$\tilde{\lambda} := \frac{\varphi(1) - \varphi(0)}{\mu \beta p}$$

and let us choose μ such that $-\lambda + \frac{\mu}{2} > 0$. We can now proceed by stating that:

$$F_\mu(t) \geq F_\mu(0) + p\left(\int_0^t \mu\beta e'(s) F_\mu(s) ds - \tilde{\lambda}\beta\mu e(t)\right) \geq F_\mu(0) + p \int_0^t \mu\beta e'(s)[F_\mu(s) - \tilde{\lambda}] ds.$$

Claim: If $F_\mu(0) \geq \tilde{\lambda}$, then $F_\mu(t) \geq \tilde{\lambda}$. It suffices to define the first time $T := \inf\{t \geq 0 : F_\mu(t) < \tilde{\lambda}\}$ and to observe from the above inequality that $T = +\infty$.

Coming back to (2.9) we get that

$$\begin{aligned} F_\mu(t) &\geq F_\mu(0) + \int_0^t \mu\Big(\beta p\, e'(s) - \lambda + \frac{\mu}{2}\Big) F_\mu(s) ds \;-\; p\,\tilde{\lambda}\beta\mu e(t) \\ &\geq F_\mu(0) + p \int_0^t \mu\beta e'(s) F_\mu(s) ds + \int_0^t \mu\Big(-\lambda + \frac{\mu}{2}\Big) F_\mu(s) ds \;-\; p\,\tilde{\lambda}\beta\mu e(t) \\ &\geq F_\mu(0) + \;p\,\tilde{\lambda}\beta\mu e(t) + \int_0^t \mu\Big(-\lambda + \frac{\mu}{2}\Big) F_\mu(s) ds - \;p\,\tilde{\lambda}\beta\mu e(t) \\ &= F_\mu(0) + \int_0^t \mu\Big(-\lambda + \frac{\mu}{2}\Big) F_\mu(s) ds. \end{aligned}$$

This implies that:

$$F_\mu(t) \geq \exp\Big(\mu\Big(-\lambda + \frac{\mu}{2}\Big)t\Big) F_\mu(0). \tag{2.10}$$

But we know that

$$F_\mu(t) \leq \exp\big(\mu\big). \tag{2.11}$$

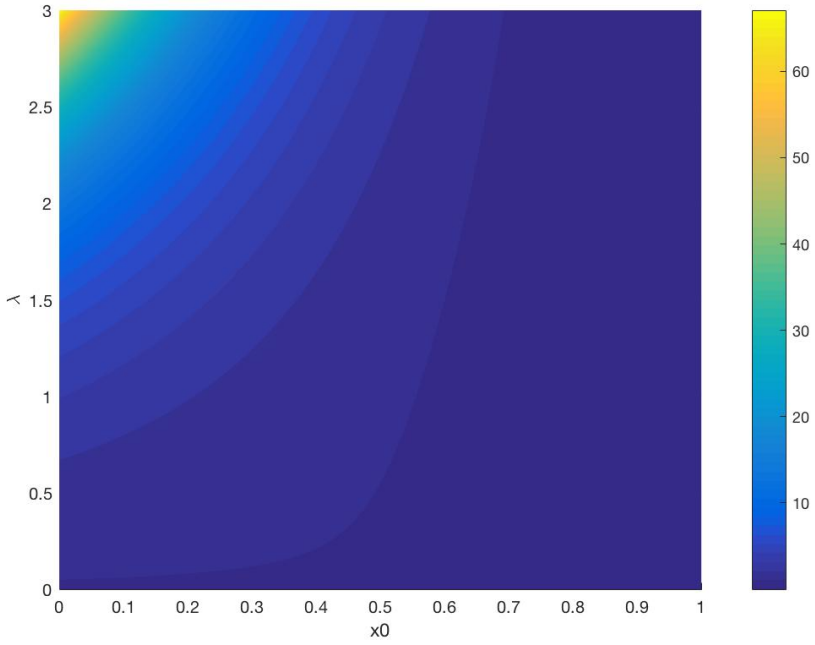


Figure 2.1: Plot of the upper bound estimate p_c^+ for the threshold p_c

From (2.10) and (2.11) we get a contradiction in the following sense: If we have a global in time solution $(X_t)_{t\geq 0}$ to (2.8) with a differentiable mean counter e and with an initial condition that satisfies $F_\mu(0) \geq \tilde{\lambda}$, then we get (2.10) and (2.11). As the latter two cannot be true at the same time, we deduce that, whenever $F_\mu(0) \geq \tilde{\lambda}$, there is no global in time solution to (2.8). □

Remark. *In summary, for a fixed (β, p) the phenomena of blow up holds for initial conditions that are concentrated around the threshold, namely for initial conditions such that*

$$\exists \mu > 2\lambda \text{ such that } F_\mu(0) \geq \tilde{\lambda} \text{ with } \tilde{\lambda} = \frac{\varphi(1) - \varphi(0)}{\mu\beta p}.$$

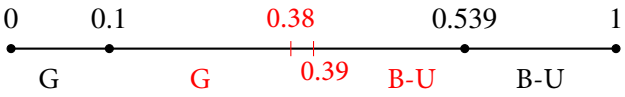
Since we assumed for simplicity that the initial condition is deterministic, we know that $F_\mu(0) = \exp(\mu x_0)$. So, we have a blow-up for all p such that:

$$p \geq \inf_{\mu \geq 2\lambda} \frac{e^\mu - 1}{\mu\beta e^{\mu x_0}} =: p_c^+(x_0)$$

In Figure 2.1 we present the upper bound for the threshold p_c^+, varying on the parameter of the drift λ and the initial condition x_0. The parameter β is taken equal to 1. Of course, values of $p_c^+(x_0)$ that are above 1 are irrelevant, in the sense that there cannot be any blow-up in those cases.

Blow up discussion for the Brownian case

Let us focus on a particular case, in absence of drift, so $\lambda = 0$, with deterministic initial condition $x_0 = 0.8$ and $\beta = 1$. From the previous remark, we get that $p_c^+ \sim 0.539$. Moreover by Theorem 5.2 in [6] we get that $p_c^- \sim 0.1$, where p_c^- is a lower bound under which there is no blow-up. From simulations, we conjecture that there exists in fact a critical threshold $p_c(x_0) \in (p_c^-(x_0), p_c^+(x_0))$ under which there is no blow-up and above which there is a blow up; numerically, the threshold is around 0.385 (in red the result obtained by numerics).



In the more general case, where $\beta \neq 1$, we get that the threshold for p is simply rescaled by the factor β.

2.2.4 Dependent Random Weights

In this section, we will present a short analysis on the behavior of the network of neurons when the synaptic weights are dependent. In particular, we will focus on two different forms of random connections.

First, we focus on the case when $\alpha^{i,j} = \alpha^i \alpha^j$, where $(\alpha^i)_{i=1,\dots,N}$ are independent Bernoulli random variables with the same parameter p, in which case the dynamics of the system reads (this is case (2) in Subsection 2.1.3):

$$X_t^i = X_0^i + \int_0^t b(X_s^i)ds + \frac{\alpha^i}{N}\sum_{j=1}^N \alpha^j M_t^j - M_t^i + W_t^i, \quad i = 1, \cdots, N. \tag{2.12}$$

Our first purpose is to find out the limit equation. We conjecture that the limit equation for this system is the following:

$$X_t = X_0 + \int_0^t b(X_s)ds + \alpha\mathbb{E}[\alpha M_t] - M_t + W_t, \tag{2.13}$$

where α is another Bernoulli random variable of parameter p and W is an independent Brownian motion. The initial condition X_0 has the same distribution as any of the X_0^i's and is independent of the pair (α, W). We guess that (2.13) is the limit equation. Importantly, (2.13) does no longer read as a rescaled version of (2.2). The reason is the processes X^i and M^i should not become independent of α^i, even if the network size tends to infinity. This is the main rationale for investigating this example.

Equation (2.13), in fact, defines a network that consists of two different components: The first corresponds to those neurons for which $\alpha = 1$; it is a complete sub-network, composed by neurons which are all connected among themselves, so that neurons of this sub-network feel an interaction with all the other neurons of the same sub-network; The second component is made of isolated neurons for which $\alpha = 0$. As such, the limit equation catches out the duality of the population and the parameter p describes the relative size of the connected sub-network within the whole population. Noticeably, neurons in the asymptotic connected sub-network evolve according to a rescaled version of (2.2), see Remark 2.2.4 right below.

The second case we would like to deal with is the one with $\alpha^{i,j} = p\alpha^i$, where again $(\alpha^i)_i$ is an i.i.d. family of Bernoulli random variables of parameter p (this is case (3) in Subsection 2.1.3). The rationale for this choice is that $\alpha^{i,j}$ here reads as the limit of $\frac{1}{N}\alpha^i \sum_{j=1}^N \alpha^j$, which prompts us below to compare this example with the previous one. In this case as well, we have a network of neurons with two different populations: one formed by isolated neurons and the other one composed by neurons that are interact with one another with the same deterministic synaptic weight p. The dynamics of the particle system reads:

$$X_t^i = X_0^i + \int_0^t b(X_s^i)ds + \frac{p\alpha^i}{N}\sum_{j=1}^N M_t^j - M_t^i + W_t^i, \quad i = 1, \cdots, N. \tag{2.14}$$

Using the fact that $p = \mathbb{E}[\alpha]$, our guess is that the limit mean field equation equation should be given by:

$$X_t = X_0 + \int_0^t b(X_s)ds + \alpha\mathbb{E}[\alpha]\mathbb{E}[M_t] - M_t + W_t. \tag{2.15}$$

Remark. *In this remark, we try to make a first comparison between* (2.13) *and* (2.15). *As for* (2.13), *we observe that*

$$\alpha\mathbb{E}\big[\alpha M_t\big] = \alpha\mathbb{E}\big[M_t \cdot \mathbb{1}_{\{\alpha=1\}}\big] = \alpha p \cdot \mathbb{E}\big[M_t|\ \alpha = 1\big],$$

which really says that, on the event $\alpha = 1$, (2.13) *behaves like* (2.4).

Regarding (2.15), *we have:*

$$\begin{aligned}\alpha p \cdot \mathbb{E}\big[M_t\big] &= \alpha p\Big(\mathbb{E}\big[M_t\big|\mathbb{1}_{\{\alpha=1\}}\big] \cdot p + \mathbb{E}\big[M_t\big|\mathbb{1}_{\{\alpha=0\}}\big] \cdot (1-p)\Big) \\ &= \alpha p^2 \cdot \mathbb{E}\Big[M_t|\ \alpha = 1\Big] + \alpha p(1-p) \cdot \mathbb{E}\Big[M_t|\ \alpha = 0\Big].\end{aligned} \tag{2.16}$$

As we expect the activity of the isolated neurons to be less than the activity of the connected ones, our guess is that the leading term in (2.16) *for causing a blow up is*

$$\alpha p^2 \cdot \mathbb{E}\Big[M_t|\ \alpha = 1\Big].$$

So if we denote by p_c^1 the critical value of p (that should depend on the initial condition) for the occurrence of a blow-up in (2.13) and similarly by p_c^2 the critical value in (2.15), we can expect that $p_c^2 \approx \sqrt{p_c^1}$. Of course, we make the conjecture, based on the final discussion of the previous section, that there is indeed, for each model, a critical value of the connectivity p under which there is no blow-up and above which there is a blow-up.

2.2.5 Derivation of the limit equations in the toy model

Following Subsection 2.2.2, we here justify the mean field term manifesting in (2.13) by focusing on a toy example. To make it clear, instead of studying rigorously the limit behavior of

$$X_t^i = X_0^i + \int_0^t b(X_s^i)ds + \frac{\alpha^i}{N}\sum_{j=1}^N \alpha^j M_t^j - M_t^i + W_t^i$$

(where α^i are i.i.d. Bernoulli variables with parameter p), we consider the simpler model introduced in Subsection 2.2.2:

$$dX_t^i = \frac{\alpha^i}{N}\sum_{j=1}^N \alpha^j X_t^j dt + dW_t^i,$$

where $X_0^i = x_0$ is deterministic. We also introduce, following the notation already used, the processes $\bar{X}^i$ described by

$$d\bar{X}_t^i = \alpha^i \mathbb{E}\left[\alpha^i \bar{X}_t^i\right]dt + dW_t^i.$$

Assuming that X^i and $\bar{X}^i$ have the same initial conditions, we get

$$X_t^i - \bar{X}_t^i = \int_0^t \frac{\alpha^i}{N}\sum_{j=1}^N \alpha^j\left(X_s^j - \bar{X}_s^j\right)ds + \int_0^t\left(\frac{\alpha^i}{N}\sum_{j=1}^N \alpha^j \bar{X}_s^j - \alpha^i\mathbb{E}\left[\alpha^i\bar{X}_s^i\right]\right)ds.$$

Taking absolute values, multiplying by α^i, summing over i and dividing by N, we can assert that

$$\begin{aligned}\frac{1}{N}\sum_{i=1}^N \alpha^i\left|X_t^i - \bar{X}_t^i\right| \le \int_0^t\left(\frac{1}{N}\sum_{i=1}^N(\alpha^i)^2\right)\left(\frac{1}{N}\sum_{i=1}^N \alpha^i\left|X_s^i - \bar{X}_s^i\right|\right)ds \\ + \int_0^t\left(\frac{1}{N}\sum_{i=1}^N(\alpha^i)^2\right)\left|\frac{1}{N}\sum_{j=1}^N \alpha^j\bar{X}_s^j - \mathbb{E}\left[\alpha^i\bar{X}_s^i\right]\right|ds.\end{aligned}$$

Defining $\tilde{\delta}_t = \frac{1}{N}\sum_{i=1}^N \alpha^i \left|X_t^i - \bar{X}_t^i\right|$, we can write that

$$\tilde{\delta}_t \leq \left(\frac{1}{N}\sum_{i=1}^N (\alpha^i)^2\right)\left[\int_0^t \tilde{\delta}_s ds + \int_0^t \left|\frac{1}{N}\sum_{j=1}^N \alpha^j \bar{X}_s^j - \mathbb{E}\left[\alpha^i \bar{X}_s^i\right]\right| ds\right].$$

By using Gronwall's lemma, we obtain

$$\tilde{\delta}_t \leq \left(\frac{1}{N}\sum_{i=1}^N (\alpha^i)^2\right)\exp\left(\frac{1}{N}\sum_{i=1}^N (\alpha^i)^2 t\right)\int_0^t \left|\frac{1}{N}\sum_{j=1}^N \alpha^j \bar{X}_s^j - \mathbb{E}\left[\alpha^i \bar{X}_s^i\right]\right| ds.$$

Also defining $\delta_t = \mathbb{E}[\tilde{\delta}_t]$, $A = \left(\frac{1}{N}\sum_{i=1}^N (\alpha^i)^2\right)$, taking expectation and using Cauchy-Schwarz inequality, we have

$$\delta_t \leq \left(\mathbb{E}\left[A^2 \exp\left(2At\right)\right]\right)^{1/2} \int_0^t \left(\mathbb{E}\left[\left(\frac{1}{N}\sum_{j=1}^N \alpha^j \bar{X}_s^j - \mathbb{E}\left[\alpha^i \bar{X}_s^i\right]\right)^2\right]\right)^{1/2} ds.$$

We can manage the last term in the expression above as we did before. In fact, using a standard independence argument, we get

$$\mathbb{E}\left[\left(\frac{1}{N}\sum_{j=1}^N \alpha^j \bar{X}_s^j - \mathbb{E}\left[\alpha^i \bar{X}_s^i\right]\right)^2\right] = \frac{1}{N}\left(\mathbb{E}\left[\left(\alpha^1 \bar{X}_s^1\right)^2\right] - \left(\mathbb{E}\left[\alpha^1 \bar{X}_s^1\right]\right)^2\right).$$

Therefore,

$$\delta_t \leq \left(\mathbb{E}\left[A^2 \exp\left(2At\right)\right]\right)^{1/2} \cdot \frac{1}{N^{1/2}} \int_0^t \left(\mathbb{E}\left[\left(\alpha^1 \bar{X}_s^1\right)^2\right] - \left(\mathbb{E}\left[\alpha^1 \bar{X}_s^1\right]\right)^2\right)^{1/2} ds,$$

and we conclude as in Subsection 2.2.2. The derivation of the toy version of equation (2.15) from the corresponding toy version of the network of interacting neurons (2.14) follows from a similar argument.

2.2.6 Blow-up Argument

We now provide analogs to Theorem 3 for (2.13) and (2.15). Let us first consider the second case, i.e., $\alpha^{i,j} = p\alpha^i$, with α^i Bernoulli independent random variables of parameter p.

Theorem 4. *Assume that drift coefficient satisfies, for some $\lambda > 0$,*

$$b(v) \geq -\lambda v, \quad \text{for all } -\infty < v \leq 1.$$

If the (deterministic) initial condition $X(0) = x_0 < 1$ is sufficiently near the threshold 1, there is no global in time solution of the SDE

$$X_t = X_0 + \int_0^t b(X_s)ds + \alpha p \cdot \mathbb{E}\big[M_t\big] + W_t - M_t,$$

where $\alpha \sim \mathscr{B}(p)$ is independent of W.

Proof. The idea of the proof is very close to that of Theorem 1. We want to find an initial condition that leads to a contradiction if we assume that $t \mapsto e(t) = \mathbb{E}(M_t)$ is differentiable. However, the proof slightly differs since we here focus on the quantity $F_\mu^\alpha(t) := \mathbb{E}[\alpha\varphi(X_t)]$. Assuming that e is indeed differentiable, applying Itô's formula to $(\varphi(X_t))_{t\geq 0}$, multiplying by α and taking the expectation, we get

$$F_\mu^\alpha(t) = F_\mu^\alpha(0) + \int_0^t \mathbb{E}\Big[\mu\alpha\varphi(X_s)\Big(-\lambda + \alpha p e'(s) + \frac{\mu}{2}\Big)\Big]ds + \Big(\varphi(0) - \varphi(1)\Big)\mathbb{E}[\alpha M_t].$$

The random variable α takes values in $\{0,1\}$ and M_t is non-negative, which makes it possible to state that $\mathbb{E}\big[\alpha M_t\big] \leq \mathbb{E}\big[M_t\big]$. Therefore, since $\big(\varphi(0) - \varphi(1)\big)$ is negative,

$$F_\mu^\alpha(t) \geq F_\mu^\alpha(0) + \int_0^t \mu\Big(-\lambda + \frac{\mu}{2}\Big)F_\mu^\alpha(s)ds + \int_0^t \mu p e'(s)F_\mu^\alpha(s)ds + \Big(\varphi(0) - \varphi(1)\Big)e(t). \quad (2.17)$$

Now, the proof exactly matches that of Theorem 1. Choose indeed $-\lambda + \frac{\mu}{2} > 0$ and take the initial condition close enough to the threshold 1 so that

$$F_\mu^\alpha(0) \leq \tilde{\lambda} \quad \text{with} \quad \tilde{\lambda} = \frac{1}{\mu p}(-\varphi(0) + \varphi(1)).$$

We then get

$$F_\mu^\alpha(t) \geq F_\mu^\alpha(0)\exp\Big(\mu\Big(-\lambda + \frac{\mu}{2}\Big)t\Big), \quad t > 0.$$

This is a contradiction because $X_t \leq 1$ and $\alpha \in \{0,1\}$, which means that $F_\mu^\alpha(t) \leq e^\mu$. □

Remark. *As explained in Remark 2.2.3, there is blow-up when*

$$\exists \mu > 2\lambda \text{ such that } F_\mu(0) \geq \tilde{\lambda} \text{ with } \tilde{\lambda} = \frac{\varphi(1) - \varphi(0)}{\mu p}.$$

Assuming for simplicity that the initial condition is deterministic, we get:

$$F_\mu^\alpha(0) = p \cdot \exp(\mu x_0)$$

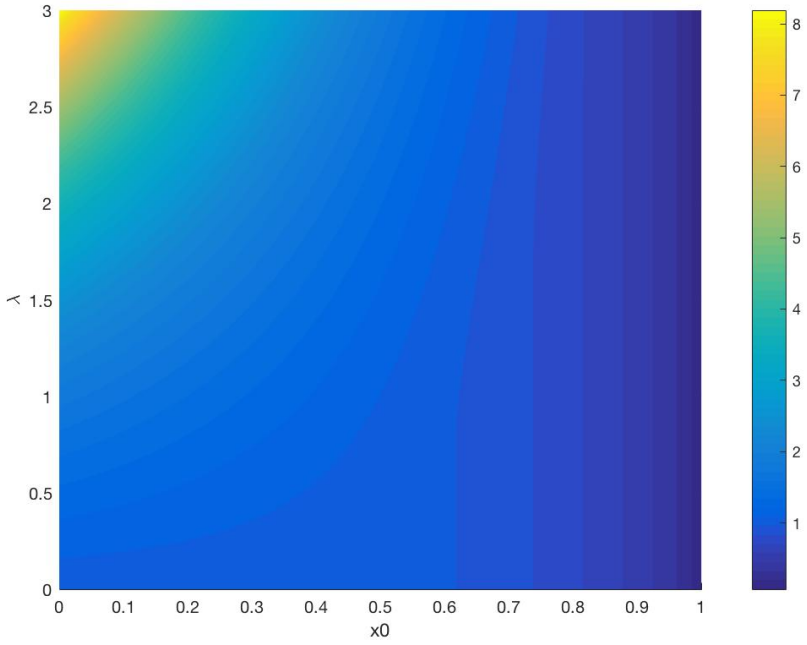


Figure 2.2: Plot of the upper bound estimate $p_c^{+,2}$ for the threshold p_c

So, given a deterministic initial condition, we have a blow-up when p is such that:

$$p \geq \sqrt{\inf_{\mu \geq 2\lambda} \frac{e^{\mu} - 1}{\mu e^{\mu x_0}}} =: p_c^{+,2}(x_0)$$

In Figure 2.2, we present the upper bound for the threshold $p_c^{+,2}$, varying on the parameter of the drift λ and the initial condition x_0.

Consider now the case $\alpha^{i,j} = \alpha^i \alpha^j$, with α^i Bernoulli independent random variables of parameter p.

Theorem 5. *Assume that drift coefficient satisfies, for some $\lambda > 0$,*

$$b(v) \geq -\lambda v, \quad \text{for all } -\infty < v \leq 1.$$

If the (deterministic) initial condition $X(0) = x_0 < 1$ is sufficiently near the threshold 1, there is no global in time solution of the SDE

$$X_t = X_0 + \int_0^t b(X_s) ds + \alpha \cdot \mathbb{E}\left[\alpha M_t\right] + W_t - M_t,$$

where $\alpha \sim \mathcal{B}(p)$ is independent of W.

Proof. The strategy is almost the same as in the previous case, namely the main tool is the function $F_\mu^\alpha(t) := \mathbb{E}[\alpha\varphi(X_t)]$. By Itô's Formula:

$$F_\mu^\alpha(t) = F_\mu^\alpha(0) + \int_0^t \mathbb{E}\Big[\mu\alpha\varphi(X_s)\Big(-\lambda + \alpha e'_\alpha(s) + \frac{\mu}{2}\Big)\Big]ds + \Big(\varphi(0) - \varphi(1)\Big)\mathbb{E}[\alpha M_t].$$

with $e_\alpha(t) = \mathbb{E}[\alpha M_t]$. With the same calculation as in the previous case, we get

$$F_\mu^\alpha(t) \geq F_\mu^\alpha(0) + \int_0^t \mu\Big(-\lambda + \frac{\mu}{2}\Big)F_\mu^\alpha(s)ds + \int_0^t \mu e'_\alpha(s)F_\mu^\alpha(s)ds + \Big(\varphi(0) - \varphi(1)\Big)e_\alpha(t).$$

The only differences with respect to the previous case are that we have $e_\alpha(t)$ instead of $e(t)$ and that there is no p in the third term. □

Remark. *As before, there is blow-up when*

$$\exists \mu > 2\lambda \text{ such that } F_\mu^\alpha(0) \geq \tilde{\lambda} \text{ with } \tilde{\lambda} = \frac{\varphi(1) - \varphi(0)}{\mu}$$

Assuming for simplicity that the initial condition is deterministic, we get:

$$F_\mu(0) = p \cdot \exp(\mu x_0).$$

So, given a deterministic initial condition, we get a blow-up when p is such that:

$$p \geq \inf_{\mu \geq 2\lambda} \frac{e^\mu - 1}{\mu e^{\mu x_0}} =: p_c^{+,1}(x_0),$$

which fits the formula obtained in Remark 2.2.3. So the upper bound varying on the parameters (x_0, λ) is the same of Figure 2.1. Observe also that

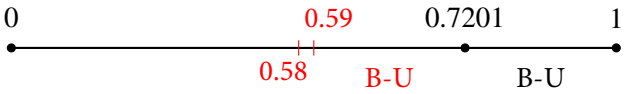
$$p_c^{+,2}(x_0) = \sqrt{p_c^{+,1}(x_0)},$$

which is consistent with our informal discussion in Remark 2.2.4.

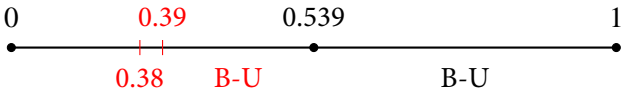
Blow up discussion for the Brownian case

Let us focus on the case where the network includes a non complete subnetwork following the dynamics prescribed in (2.15) (or equivalently case (3) in Subsection 2.1.3) in absence of drift, i.e. $b \equiv 0$, with deterministic initial condition $x_0 = 0.8$. From the previous remark, we get

that $p_c^{+,2} \sim 0.7201$. Through simulations, we look for the threshold value p_c (with the same conjecture as before that, above this threshold, there is a blow-up whilst there is no blow-up below the threshold). Here is the plot that we get (numerical values are in red):



Recall from the discussion right after Remark 2.2.3 the plot we get for the network of type (2.13) (or equivalently case (2) in Subsection 2.1.3):



Notice that the numerical results confirm what we predict in Remarks 2.2.4, 2.2.6, 2.2.6; observe in particular that $\sqrt{0.39} \sim 0.62$, which not so far from 0.59.

2.2.7 Comparison between the two models

In this section we continue to investigate the differences between the two cases (2) and (3) in Subsection 2.1.3. We here consider the following SDEs:

$$X_t^i = X_0^i + \int_0^t b(X_s^i)ds + \frac{\alpha^i}{N} p \sum_{j=1}^N M_t^j + W_t^i - M_t^i,$$
$$X_t^i = X_0^i + \int_0^t b(X_s^i)ds + \frac{\alpha^i}{N} \sum_{j=1}^N \alpha^j M_t^j + W_t^i - M_t^i,$$

with the usual definitions. Recall that our guess is that the two systems converge to the limit SDEs (compare with (2.13) and (2.15)):

$$X_t = X_0 + \int_0^t b(X_s)ds + \alpha p \mathbb{E}\big[M_t\big] + W_t - M_t,$$
$$X_t = X_0 + \int_0^t b(X_s)ds + \alpha \mathbb{E}\big[\alpha M_t\big] + W_t - M_t,$$

again with the usual definitions. Using the same conditioning argument as in Remark 2.2.4, these two last SDEs can easily be rewritten as:

$$X_t = X_0 + \int_0^t b(X_s)ds + \alpha p\Big(p \cdot \mathbb{E}\big[M_t \big| \alpha = 1\big] + (1-p) \cdot \mathbb{E}\big[M_t \big| \alpha = 0\big]\Big) + W_t - M_t,$$
$$X_t = X_0 + \int_0^t b(X_s)ds + \alpha\Big(p \cdot \mathbb{E}\big[M_t \big| \alpha = 1\big]\Big) + W_t - M_t.$$

Assuming, as we did to compute thresholds numerically, that the initial condition is deterministic, i.e., $X_0 = x_0$, and that b is zero, we finally get:

$$X_t = x_0 + \Big(\alpha p^2 \cdot \mathbb{E}\big[M_t \big|\, \alpha = 1\big] + \alpha p(1-p) \cdot \mathbb{E}\big[M_t \big|\, \alpha = 0\big]\Big) + W_t - M_t, \tag{2.18}$$

$$X_t = x_0 + \Big(\alpha p \cdot \mathbb{E}\big[M_t \big|\, \alpha = 1\big]\Big) + W_t - M_t. \tag{2.19}$$

We now study equation (2.18) and compare it to (2.19). First, let us consider the term $\mathbb{E}\big[M_t \big|\, \alpha = 0\big]$: this is the expectation of the process M_t given the fact that the neuron is not concerned by interaction. The good point is that the limiting model without interaction can be solved explicitly. Namely, we can identify $\mathbb{E}\big[M_t \big|\, \alpha = 0\big]$ with $\mathbb{E}\big[\tilde{M}_t\big]$, where $\tilde{M}_t$ solves the SDE:

$$\tilde{X}_t + \tilde{M}_t = x_0 + \tilde{W}_t, \qquad \tilde{M}_t = \left\lfloor \left(\sup_{s\in[0,t]} \big(\tilde{X}_s + \tilde{M}_s\big)\right)_+ \right\rfloor = \left\lfloor \left(x_0 + \sup_{s\in[0,t]} \tilde{W}_s\right)_+ \right\rfloor,$$

where $\lfloor y \rfloor$ is the floor part of y. Calling f_t the density of the running maximum of Brownian motion until time t, we get:

$$\mathbb{E}\big[\tilde{M}_t\big] = \int_{\mathbb{R}} \Big\lfloor \big(x_0 + x\big)_+ \Big\rfloor f_t(x) dx.$$

Recalling that $f_t(x) = \big[\sqrt{2/\pi t}\, \exp\big(-x^2/2t\big)\big] \mathbb{1}_{\{x\geq 0\}}$, the above expression becomes

$$\begin{aligned}
\mathbb{E}\big[\tilde{M}_t\big] &= \sqrt{\frac{2}{\pi t}} \int_{\mathbb{R}^+} \Big\lfloor \big(x_0 + x\big)_+ \Big\rfloor \exp\left(-\frac{x^2}{2t}\right) dx \\
&= \sqrt{\frac{2}{\pi t}} \int_{-x_0 \vee 0}^{+\infty} \Big\lfloor \big(x_0 + x\big)_+ \Big\rfloor \exp\left(-\frac{x^2}{2t}\right) dx \\
&= \frac{4}{\sqrt{\pi}} \int_{\left(-x_0/\sqrt{2t}\right)\vee 0}^{+\infty} \Big\lfloor \left(x_0 + \sqrt{2t}x\right)_+ \Big\rfloor \exp\big(-x^2\big) dx \\
&= \frac{4}{\sqrt{\pi}} \sum_{k\in\mathbb{N}} k \int_{\left(-x_0+k\right)/\sqrt{2t}\big)\vee 0}^{\left((-x_0+k+1)/\sqrt{2t}\right)\vee 0} \exp\big(-x^2\big) dx.
\end{aligned}$$

Particularly, since $x_0 < 1$, it holds that $-x_0 + k$ is always positive for $k \geq 1$: therefore, $(-x_0 + k)/\sqrt{2t} \vee 0 = (-x_0 + k)/\sqrt{2t}$. Calling $e_0(t) := \mathbb{E}[\tilde{M}_t]$, we then see that the function e_0 is differentiable and its derivative is

$$e_0'(t) = \frac{1}{t}\sqrt{\frac{2}{\pi t}} e^{-\frac{x_0^2}{2t}} \sum_{k\in\mathbb{N}} k \left[(-x_0 + k) e^{\frac{-k^2 + 2x_0 k}{2t}} - (-x_0 + k + 1) e^{\frac{-(k+1)^2 + 2x_0(k+1)}{2t}} \right].$$

Since $x_0 < 1$, we have $-k^2 + 2x_0 k \geq 0$ whenever $k \geq 2$. Hence, we obtain the following:

$$e_0'(t) \leq \frac{1}{t}\sqrt{\frac{2}{\pi t}}e^{-\frac{x_0^2}{2t}}\left[(1-x_0)\left[e^{\frac{-1+2x_0}{2t}} - e^{\frac{-2(1-x_0)}{t}}\right] + \right.$$
$$\left. + \sum_{k\geq 2} k\left[(-x_0+k)e^{\frac{-k^2+2x_0k}{2t}} - (-x_0+k+1)e^{\frac{-(k+1)^2+2x_0(k+1)}{2t}}\right]\right],$$

and then:

$$e_0'(t) \leq \frac{1}{t}\sqrt{\frac{2}{\pi t}}(1-x_0)e^{-\frac{(1-x_0)^2}{2t}} + \frac{1}{t}\sqrt{\frac{2}{\pi t}}e^{-\frac{x_0^2}{2t}}\sum_{k\geq 2} k\left[(-x_0+k)e^{\frac{-k^2+2x_0k}{2T}} - (-x_0+k+1)e^{\frac{-(k+1)^2+2x_0(k+1)}{2T}}\right],$$

that is, for a finite constant $C_{x_0,T}$,

$$\sup_{0\leq t\leq T} e_0'(t) \leq C_{x_0,T}.$$

We now come back to (2.18) and (2.19). We can rewrite (2.18) in the form:

$$X_t = x_0 + \alpha p(1-p)\int_0^t e_0'(s)ds + \left(\alpha p^2 \cdot \mathbb{E}\left[M_t \middle| \alpha = 1\right]\right) + W_t - M_t, \tag{2.20}$$

which shows that, in comparison with (2.19), there is not only an additional factor p in the conditional mean field term given the event $\alpha = 1$, but there is also an additional drift e_0'. In other words, if we omit the term $\alpha p(1-p)\int_0^t e_0'(s)ds$ in (2.20), we recover (2.19), but with p replaced by p^2. Intuitively, this says that, if e_0' was equal to 0, we would expect $p_c^2 = \sqrt{p_c^1}$. In fact, e_0' is positive: It helps pushing the particles towards the threshold. As a result, it makes sense to expect $p_c^2 < \sqrt{p_c^1}$. Fortunately, this is exactly what shows up in the numerical analysis performed at the end of Subsection 2.2.6.

2.2.8 Independent Bernoulli random variables with parameter p_N

We now address case (4) in Subsection 2.1.3, namely we deal with a neural network described by the equation

$$X_t^i = X_0^i + \int_0^t b(X_s^i)ds + \frac{\beta}{p_N \cdot N}\sum_{j=1}^N \alpha_N^{i,j} M_t^j + W_t^i - M_t^i,$$

where $\alpha_N^{i,j}$ are i.i.d. Bernoulli random variables with parameter p_N **depending** on N, the total number of neurons in the network. We are interested in three cases:

- $p_N \cdot N = \log^{1/2}(N)$;

- $p_N \cdot N = \log(N)$;

- $p_N \cdot N = \log^2(N)$.

The reason why we choose these three parameters follows from the theory of Erdös-Renyi for random graphs, see [9, 10] for undirected graphs and [13, 22] for directed graphs. We indeed know that $p_N = \log(N)/N$ is a sharp threshold for connectedness, meaning that:

- If $p_N < (1-\varepsilon)\log(N)/N$, then the probability that the graph has a unique connected component tends to 0 as N goes to infinity;

- $p_N > (1+\varepsilon)\log(N)/N$, then the probability that the graph has a unique connected component tends to 1 as N goes to infinity.

In contrast with more standard particle systems like those addressed in [23] or (say) of the same form as in (2.5), it is here possible to have several neurons hitting the firing potential at the same instant of time, and, most of all, it is even possible to have a neuron spiking more than once at the same instant of time, which fact may be excluded in other models of neural networks, see for instance [6]. As exemplified in [4, 7], this requires a sequential definition of the spikes that may occur at the same time, given by an induction with the following initialization:

$$\Gamma_0 := \left\{ i \in \{1, \cdots, N\} \,:\, X^i_{t-} = 1 \right\}.$$

We say that a spike occurs at time t if $\Gamma_0 \neq \varnothing$. Because of the interactions between the neurons, neurons in the set Γ_0 may force the others to jump at the same time t. This happens for neuron $i \notin \Gamma_0$ if

$$X^i_{1,t-} := X^i_{t-} + \frac{\beta}{p_N \cdot N} \sum_{j=1}^{N} \alpha_N^{i,j} \mathbb{1}_{\{j \in \Gamma_0\}} \geq 1,$$

which prompts us to let $\Gamma_1 := \{i \in \{1, \cdots, N\} \setminus \Gamma_0 \,:\, X^i_{1,t-} \geq 1\}$. Iteratively, we define for any $i \in \{1, \cdots, N\}$:

$$X^i_{k+1,t-} := \begin{cases} X^i_{k,t-} + \dfrac{\beta}{p_N \cdot N} \displaystyle\sum_{j=1}^{N} \alpha_N^{i,j} \mathbb{1}_{\{j \in \Gamma_k\}} & \text{if } i \notin \Gamma_k, \\ X^i_{k,t-} - \lfloor X^i_{k,t-} \rfloor & \text{if } i \in \Gamma_k, \end{cases}$$
$$\Gamma_{k+1} := \left\{ i \in \{1, \ldots, N\} \backslash \Gamma_k \,:\, X^i_{k+1,t-} \geq 1 \right\}.$$

And then we let

$$\begin{aligned} M_t^i - M_{t-}^i &= \sum_{k\in\mathbb{N}} \mathbb{1}_{\Gamma_k}(i), \\ X_t^i &= \lim_{k\to\infty} X_{k,t-}^i \quad \text{if the limit exists.} \end{aligned} \tag{2.21}$$

Intuitively, Γ_k is the set of neurons that spike at the kth iteration, $X_{k,t-}^i$ is the potential of neuron i before the kth iteration and $X_{k+1,t-}^i$ is the potential after the kth iteration. With this rule we observe that:

- It is possible that one neuron receives (at a given iteration) a cumulative kick (from all the other particles) that is greater than 1, meaning that the potential of a neuron can jump from a potential less than 1 to a potential greater than 2. With our definition of $X_{k+1,t-}^i$ if $i \in \Gamma_k$, we just regard the whole as a single spike.

Still a neuron can spike several times at the same time:

- It is indeed possible that a neuron i spikes, that its kick makes others spike and that those, in return, make i spike again. This behavior may repeat again: when it repeats just for a finite number of times, we call it a "finite cascade". In that case, the limit in (2.21) is well defined.
- It may happen that the cascading behavior just described above goes on infinitely many times: we call it an "infinite cascade". In that case the limit in (2.21) may not exist.

Below, we choose $\beta < 1$. We then try to make a connection between neurons that spike more than twice and neurons that have a large degree, where, by definition, the degree d_i of neuron i is $d_i := \sum_{j=1}^N \alpha_N^{i,j}$. Our guess is based on the fact that, for a neuron that is connected to neurons that do not spike more than once at time t, the only way for it to record more than two spikes at time t is that its degree is higher than $\lceil p_N N/\beta \rceil$. In particular, the first neurons that record a second spike in the inductive construction of the sets $(\Gamma_k)_k$ must have a large degree. In this regard, we show below that the probability that a neuron has a high degree gets smaller and smaller as N tends to ∞. We thus conjecture that, for the prescribed values of parameters, most of the neurons can only jump once at a given time and, henceforth, that, among those that record more than two spikes at the same time, most of them have a large degree.

We are therefore interested in computing

$$\mathbb{P}\left[\frac{1}{N}\sum_{j=1}^N \alpha_N^{1,j} \geq \frac{p_N}{\beta}\right],$$

where the apex 1 may be substituted by any other index i.

Remark (Estimate on the upper bound for the probability to have a multiple spikes)**.** *Observing that p_N/β is bigger than p_N and using the Cramer's Theorem (in the context of Large Deviation Theory), we find out that*

$$\mathbb{P}\left[\frac{1}{N}\sum_{j=1}^{N}\alpha_N^{1,j} \geq \frac{p_N}{\beta}\right] \leq e^{-N\cdot\Lambda^*\left(\frac{p_N}{\beta}\right)}, \tag{2.22}$$

where

$$\Lambda^*(x) := x\log\left(\frac{x}{p_N}\right) + (1-x)\log\left(\frac{1-x}{1-p_N}\right).$$

We have that

$$-\log\left(e^{-N\cdot\Lambda^*\left(\frac{p_N}{\beta}\right)}\right) = \frac{1}{\beta}\log\left(\frac{1}{\beta}\right)\log^k(N) + \left(N - \frac{\log^k(N)}{\beta}\right)\log\left(\frac{\beta N - \log^k(N)}{\beta N - \beta\log^k(N)}\right), \tag{2.23}$$

where k is such that $p_N = \log^k(N)/N$ (recall that $k = 1/2, 1, 2$ in the three typical cases we have in mind). The last part above is

$$\left(N - \frac{\log^k(N)}{\beta}\right)\log\left(\frac{\beta N - \log^k(N)}{\beta N - \beta\log^k(N)}\right) = \left(N - \frac{\log^k(N)}{\beta}\right)\log\left(1 - \frac{1-\beta}{\beta}\frac{\log^k(N)}{N - \log^k(N)}\right).$$

Therefore, the last expression becomes

$$\left(N - \frac{\log^k(N)}{\beta}\right)\left(-\frac{1-\beta}{\beta}\frac{\log^k(N)}{N-\log^k(N)} + \mathcal{O}\left(\left(\frac{\log^k(N)}{N-\log^k(N)}\right)^2\right)\right),$$

which can be rewritten as

$$\begin{aligned}
&\frac{\beta N - \log^k(N)}{\beta}\left(-\frac{(1-\beta)\log^k(N)}{\beta\left(N-\log^k(N)\right)}\right) + \frac{\beta N - \log^k(N)}{\beta}\cdot\mathcal{O}\left(\left(\frac{\log^k(N)}{N-\log^k(N)}\right)^2\right)\\
&= \frac{\beta N - \log^k(N)}{\beta N - \beta\log^k(N)}\left(-\frac{(1-\beta)\log^k(N)}{\beta}\right)\\
&\quad + \frac{\beta N - \log^k(N)}{\beta N - \beta\log^k(N)}\cdot\frac{N-\log^k(N)}{\log^k(N)}\log^k(N)\cdot\mathcal{O}\left(\left(\frac{\log^k(N)}{N-\log^k(N)}\right)^2\right).
\end{aligned}$$

Finally, we can again rewrite the expression above as

$$\left(\frac{\beta N - \log^k(N)}{\beta N - \beta\log^k(N)}\right)\left(-\frac{1-\beta}{\beta} + \mathcal{O}\left(\frac{\log^k(N)}{N - \log^k(N)}\right)\right)\log^k(N).$$

Now, the first term in the expression above converges to 1 *when* $N \to +\infty$*; similarly, the "big O" multiplied by* $\log^k(N)$ *goes to* 0 *when* $N \to +\infty$*; this means that, for every* $c < 1$*,*

$$\left(\frac{\beta N - \log^k(N)}{\beta N - \beta\log^k(N)}\right)\left(-\frac{1-\beta}{\beta} + \mathcal{O}\left(\frac{\log^k(N)}{N - \log^k(N)}\right)\right)\log^k(N) \geq -c\left(\frac{1-\beta}{\beta}\right)\log^k(N),$$

for N large enough. Coming back to (2.23):

$$-\log\left(e^{-N\cdot\Lambda^*\left(\frac{p_N}{\beta}\right)}\right) \geq \left[\frac{1}{\beta}\left(\log\left(\frac{1}{\beta}\right) - c\right) + c\right]\log^k(N).$$

Calling $C(c,\beta) := \frac{1}{\beta}\left(\log\left(\frac{1}{\beta}\right) - c\right) + c > \frac{1}{\beta} - 1 - \frac{c}{\beta} + c = (\frac{1}{\beta} - 1)(1-c) > 0$*, we deduce that*

$$e^{-N\cdot\Lambda^*\left(\frac{p_N}{\beta}\right)} \leq e^{-C(c,\beta)\log^k(N)},$$

which is especially meaningful when $k > 0$*. To conclude, we have found that, for N large enough,*

$$\mathbb{P}\left[\frac{1}{N}\sum_{j=1}^{N}\alpha_N^{1,j} \geq \frac{p_N}{\beta}\right] \leq e^{-C(c,\beta)\log^k(N)}. \tag{2.24}$$

Our conjecture is that, for $k > 0$*, this should be a good approximation of the probability to observe more than two spikes for a single neuron at the same time.*

By letting c tend to 1 in (2.24), we get in fact the following large deviation upper bound.

Theorem 6. *Assume that* $\beta < 1$ *and choose* p_N *in the form* $p_N = \log^k(N)/N$*. Then,*

$$\limsup_{N\to\infty} a_N^{-1}\log\left(\mathbb{P}\left[\frac{1}{N}\sum_{j=1}^{N}\alpha_N^{1,j} \geq \frac{p_N}{\beta}\right]\right) \leq -C(\beta), \tag{2.25}$$

with $a_N = N\cdot p_N = \log^k(N)$ *and* $C(\beta) := \frac{1}{\beta}\left(\log\left(\frac{1}{\beta}\right) - 1\right) + 1$*.*

Remark. *As for the lower bound, we did not manage to implement the usual tilting method used in the proof of Cramer's theorem. The fact that* $(a_N)_{N\geq 0}$ *increases at a rate that is much smaller*

than the rate of convergence of the law of large numbers, as given by the central limit theorem, is a hindrance.

2.2.9 Comparison with the Mean Field Case

A crucial question is to decide whether the mean field approximation is still valid in each of the three cases addressed in the preliminary discussion of Subsection 2.2.8. Observe indeed that, since p_N is assumed to get smaller and smaller as N tends to 0, it is by no means clear that the arguments used in Subsections 2.2.2 and 2.2.5 still apply. So, we consider the network given by a family of i.i.d. Bernoulli synaptic weights $(\alpha_N^{i,j})_{i,j} \sim \mathscr{B}(p_N)$, with p_N as before, and we want to compare numerically the behavior of the particle system driven by the interaction term

$$\frac{\beta}{N \cdot p_N} \sum_{j=1}^{N} \alpha_N^{i,j} M_t^j \tag{2.26}$$

with the behavior of the particle system driven by the interaction term

$$\frac{1}{N} \sum_{j=1}^{N} \gamma^{i,j} M_t^j, \tag{2.27}$$

where $(\gamma^{i,j})_{i,j} \sim \mathscr{B}(\beta)$ are i.i.d. Bernoulli variables of parameter β. The comparison between both is motivated by the fact that $\mathbb{E}[\beta \alpha_N^{i,j}/p_N] = \beta$. And for sure, our preliminary investigations have shown that (2.27) had the same limit behavior as the model driven by the simpler interaction term

$$\frac{\beta}{N} \sum_{j=1}^{N} M_t^j. \tag{2.28}$$

We provide in the left pane in Figure 2.3) simulations of the interaction terms in both cases, (2.26) and (2.27), when $p_N = \log^2(N)/N$ and $\beta = 0.375$. It shows that, for N large, both are indeed close. This seems to be a numerical evidence that the mean field approximation should be valid when p_N is above the Erdös-Renyi threshold for connectedness of the interaction graph. When $p_N = \log(N)/N$ and $p_N = \log^{1/2}(N)/N$, and for the same value of β, the center and right panes in Figure 2.3 suggest that the mean field approximation is no longer valid below the Erdös-Renyi threshold. These observations seem to be in accordance with the results obtained in [8] for the mean field approximation of another interacting diffusion model (so-called Kuramoto model) on a random Erdös-Renyi graph: Therein, the mean field approximation is shown to hold true if $p_N N/\log(N) \to \infty$.

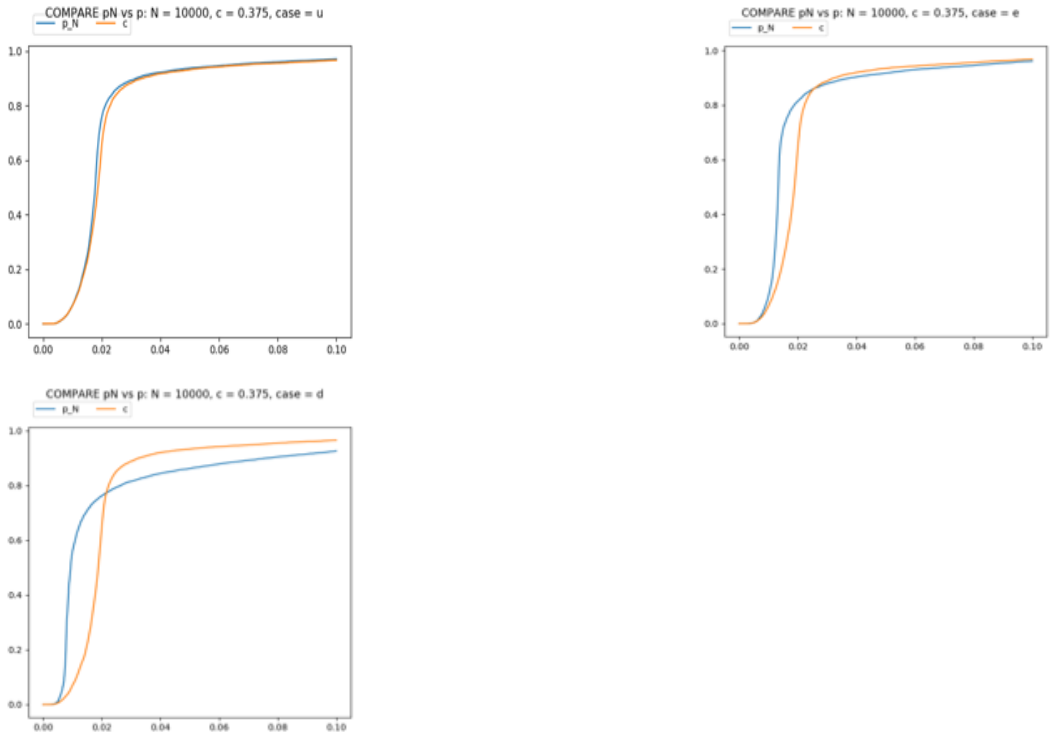


Figure 2.3: Comparison between interactions terms for $p_N = \log^2(N)/N$, $p_N = \log(N)/N$ and $p_N = \log^{1/2}(N)/N$.

2.3 Algorithm

2.3.1 Introduction

In this section, we present the algorithms used to simulate the particles system (2.3). We also consider a slightly different model in Section 2.3.3, where the neurons no longer spike when their membrane potential crosses a fixed deterministic threshold but they spike with *rate* $f^i(X_t^i)$ at time t. (see Section 2.3.3 for a precise definition).

Clock-driven vs. event-driven algorithm

The simulation of a very large network of neurons requires to carefully organize the program. We have used two different methods.

Clock-driven method. We fix a time step Δ and approximate the values of the membrane potentials at each time step $T_0 + \ell\Delta$ for $\ell \in \mathbb{N}$. This method is quite simple to implement,

but not very fast. The choice of the time step Δ is crucial: it has to be very small compared to the typical length of an InterSpike Interval (ISI).

Event-driven method. This method consists in, first, simulating the times at which *events* occur, and second, updating the state of the network at these times. The advantage is that the clock is automatically fitted to the state of the system. In our algorithm, we have chosen the spiking times to describe the important *events* of this method.

We have used the *clock-driven method* to simulate the hard threshold model and the *event-driven method* for the soft threshold model. In this case, our algorithm is an extension of Ogata's algorithm, see [20].

2.3.2 Hard Threshold

We consider the finite system defined by (2.1), which we rewrite below for the sake of convenience:

$$\begin{cases} X_t^i = X_0^i + \displaystyle\int_0^t b(X_s^i)ds + \frac{\beta}{N}\sum_{i=1}^N \alpha^{i,j}M_t^j + W_t^i - (X_F - X_R)M_t^i, \\ M_t^i = \displaystyle\sum_{k\in\mathbb{N}} \mathbb{1}_{[0,t]}(\tau_k^i), \end{cases} \qquad \forall i \in \Big\{1, ..., N\Big\}, \tag{2.29}$$

where, as usual, τ_k^i represents the time at which neuron i reaches the threshold $X_F > 0$ for the k-th time; above, X_R is the (common) reset values of the neurons.

In this section, we simulate the network at each time step $\ell\Delta$, where Δ is our fixed discretization parameter. The network is fully described by $((\bar{X}^i_{\ell\Delta}, \bar{M}^i_{\ell\Delta}), i \in \{1, \dots, N\}, \ell\Delta \le T)$.

- Initialization $\boldsymbol{\ell = 0}$.
  - We set $\bar{M}_0^i = 0$ and simulate the membrane potentials $\bar{X}_0^i$ as independent and identically distributed according to the initial law.
- Step $\boldsymbol{\ell \to (\ell + 1)}$.
  - Evolution of the membrane potentials without interaction, nor spontaneous spikes:

$$\hat{\bar{X}}^i_{(\ell+1)\Delta} = \bar{X}^i_{\ell\Delta} + b(\bar{X}^i_{\ell\Delta})\Delta + W^i_{(\ell+1)\Delta} - W^i_{\ell\Delta}.$$

  - Construction of the set $\Gamma^0_{(\ell+1)\Delta}$ of those neurons that are spiking first at time $(\ell+1)\Delta$. First, set $\Gamma^0_{(\ell+1)\Delta} := \{i \in \{1, \dots, N\}, \hat{\bar{X}}^i_{(\ell+1)\Delta} \ge X_F\}$. For each neuron $i \notin \Gamma^0_{(\ell+1)\Delta}$, we have $\bar{X}^i_{\ell\Delta} \le X_F$ and $\hat{\bar{X}}^i_{(\ell+1)\Delta} \le X_F$ but it could happen that $\sup_{\ell\Delta \le t \le (\ell+1)\Delta} \hat{\bar{X}}^i_t \ge X_F$, i.e. the process crosses the threshold during one time step but is back below the

threshold at time $(\ell+1)\Delta$. We approximate the probability of such an event by the well known corresponding probability for a Brownian bridge (see [12]), that is

$$p^i = \exp\left(-2\frac{(X_F - \hat{\bar{X}}^i_{(\ell+1)\Delta})(X_F - \bar{X}^i_{\ell\Delta})}{\Delta}\right).$$

More precisely, for each $i \notin \Gamma^0_{(\ell+1)\Delta}$, we simulate $U^i_{\ell+1}$, uniformly distributed on $[0,1]$, independently of everything. If $U^i_{\ell+1} < p^i$, we add the neuron i in the set $\Gamma^0_{(\ell+1)\Delta}$ of spiking neurons.

– Construction of the cascade. Following Subsection 2.2.8, we let for $i \notin \Gamma^0_{(\ell+1)\Delta}$:

$$\hat{\bar{X}}^i_{1,(\ell+1)\Delta} := \hat{\bar{X}}^i_{(\ell+1)\Delta} + \sum_{j\in\Gamma^0_{(\ell+1)\Delta}} J_N^{j\to i},$$

and then $\Gamma^1_{(\ell+1)\Delta} := \{i \in \{1,\cdots,N\} \setminus \Gamma^0_{(\ell+1)\Delta} : \hat{\bar{X}}^i_{1,(\ell+1)\Delta} \geq X_F\}$. Iteratively, we define for any $i \in \{1,\cdots,N\}$:

$$\hat{\bar{X}}^i_{k+1,(\ell+1)\Delta} := \begin{cases} \hat{\bar{X}}^i_{k,(\ell+1)\Delta} + \displaystyle\sum_{j\in\Gamma^k_{(\ell+1)\Delta}} J_N^{j\to i} & \text{if } i \notin \Gamma^k_{(\ell+1)\Delta}, \\ \hat{\bar{X}}^i_{k,(\ell+1)\Delta} - \left\lfloor \dfrac{\hat{\bar{X}}^i_{k,(\ell+1)\Delta} - X_R}{X_F - X_R} \right\rfloor (X_F - X_R) & \text{if } i \in \Gamma^k_{(\ell+1)\Delta}, \end{cases}$$

$$\Gamma^{k+1}_{(\ell+1)\Delta} := \left\{ i \in \{1,\dots,N\} \setminus \Gamma^k_{(\ell+1)\Delta} : \hat{\bar{X}}^i_{k+1,(\ell+1)\Delta} \geq X_F \right\}.$$

– Final update. Call $k_{\max}$ the first index when $\Gamma^{k_{\max}}_{(\ell+1)\Delta} = \varnothing$. And then we let

$$\bar{M}^i_{(\ell+1)\Delta} = \bar{M}^i_{\ell\Delta} + \sum_{k=0}^{k_{\max}-1} \mathbb{1}_{\Gamma^k_{(\ell+1)\Delta}}(i),$$
$$\bar{X}^i_{(\ell+1)\Delta} = \hat{\bar{X}}^i_{k_{\max},(\ell+1)\Delta}.$$

Algorithm 8 Pseudo code for the simulation of a network of interacting neurons in case of hard threshold

Initialize the processes X^i and M^i of neuron i, for each $i = 1, ..., N$;
repeat
simulate the potential of each neuron for one time-step not considering interaction or resetting;
add every neuron with potential above the threshold to the list of spiking neurons;
for i neuron still below the threshold **do**
simulate the event "a Brownian bridge crosses the threshold" for neuron i;
update the list of spiking neurons;
end for
repeat
update the entire network by adding kicks caused by spiking;
reset the spiking neurons;
update the list of spiking neurons;
until the cascade of spiking neurons has been exhausted
until we have simulated the network for enough time-steps

2.3.3 Soft threshold

We now address a new model, which leads to a new algorithm. We explain below how this new model is connected with the previous, but, first, we want to make clear the reasons why we introduce this new model:

- One issue with the model developed previously is that it allows a cascade phenomenon; this turns out to be a hindrance from the numerical point of view. We are thus interested in having a smoother version.
- Anyway, the model studied up to now is not perfectly adapted to biological measurements: one main criticism (to which we already alluded in introduction) is precisely the fact that spikes are transmitted instantaneously to the post-synaptic neurons. Indeed, in (2.3), the post-synaptic neuron j receives a kick of size $\boldsymbol{J}_N^{i\to j}$ exactly at the spiking time of the pre-synaptic neuron i. If its membrane potential exceeds the threshold, the neuron j also spikes at the same time.

The idea for solving this issue is to use a point process based model (see [5, 11] for related references), with the following main features: Neurons have a probability of spiking that is strictly less than one when their potential reaches the threshold (which explains why the threshold is no

longer *hard*). Also the probability of having two spikes at the exact same time is zero (as long as the potential has a finite value) and so there is no more possibility of a cascade. A point process based system can be difficult to simulate, but there exist recipes for simplifying the creation of the algorithm. One of them is an algorithm developed by Ogata, see [20], specifically designed for simulating point processes. We make it clear below.

Our soft threshold model is a Markov process based upon the following general rule. The neuron i spikes at time t with rate $f^i(X_t^i)$, that is

$$\lim_{dt\to 0} \frac{1}{dt}\mathbb{P}\big(\text{neuron } i \text{ spikes in } [t, t+dt] \mid \sigma(X_s^i, s < t, i = 1, \cdots, N)\big) = f^i(X_t^i).$$

Also, given the history before t, neurons behave independently until a new neuron spikes. In particular, for such a model, two neurons spike precisely at the same instant with probability 0, which explains why we have no more cascades. Between two spikes in the system, the membrane potentials of the neurons evolve independently according to the same diffusive dynamics as in (2.1); when a neuron spikes, all the membrane potentials receive a kick, as prescribed by the values of the synaptic weights.

Remark. 1. *One can think of the hard threshold model as a particular case of the soft threshold model with the rate functions*

$$f^i(x) = \begin{cases} +\infty \text{ if } x \geq X_F \\ 0 \text{ if } x < X_F, \end{cases}$$

in which case the rate functions are the same for all the neurons.

2. *the choice $f^i(x) = C\big((x - X_F)_+\big)^p$ with a large $p \in \mathbb{N}$ and a good choice of constant C should be a good approximation of the hard threshold model. Notice that this intensity function is equal to zero for values of the potential that are below X_F, the latter still playing the role of a threshold: biological neurons indeed do not spike when their potential is below some threshold (actually it depends on the type of neurons we are dealing with, but this is mostly true for cortex neurons). When the potential exceeds X_F, the probability of spiking should quickly increase with the membrane potential so that the neuron will quickly spike.*

The soft threshold model can be rigorously defined as a multivariate point process. Although we do not provide the definition explicitly, we borrow materials from this theory to construct our algorithm.

- First, we know that, given the membrane potentials $X_t^1, \cdots, X_t^N$ at a time t (t being typically an event of the multivariate point process), the first next event $t + T$ after t (or equivalently,

the first time after t at which a new neuron spikes) is given by the first occurrence of a new point in a point process of intensity $(\sum_{i=1}^N f^i(X^i_{t+\theta}))_{\theta\geq 0}$.

- Second, given T, the label i of the spiking neuron is distributed proportionally to $f^i(X^i_{t+T})$.

Although this construction is pretty simple, it is not possible to implement it directly in the form of a simulation method. So, in practice, we combine the last two points with a rejection procedure. Ideally, it reads as follows:

- For all, $i = 1, \dots, N$, we introduce a (predictable) process $(\tilde{f}^i_{t+\theta})_{\theta\geq 0}$ such that $f^i(X^i_{t+\theta}) \leq \tilde{f}^i_{t+\theta}$, for all $\theta \geq 0$;
- provided we can do so, we simulate, conditional on the history up until time t, the next event (or next point) $t + T$ of a point process with intensity $(\sum_{i=1}^N \tilde{f}^i_{t+\theta})_{\theta\geq 0}$;
- we choose the label i_0 of the spiking neuron proportionally to $\tilde{f}^{i_0}_{t+T}$;
- we simulate the membrane potential $X^{i_0}_{t+T}$ according to the sole diffusive dynamics prescribed before;
- we *accept* the spike of neuron i_0 at time $t + T$ with probability $f^i(X^{i_0}_{t+T})/\tilde{f}^{i_0}_{t+T}$. In this case, we update its membrane potential to its reset value $X^{i_0}_{t+T} = X_R$ and add the kick $J_N^{i_0\to j}$ to the post-synaptic neurons j (i.e. such that $J_N^{i_0\to j} \neq 0$);
- we then restart the procedure.

Of course, the choice of the "approximation functions" $(\tilde{f}^i_{t+\theta})_{\theta\geq 0}$ is crucial, as the next event of the process $(\sum_{i=1}^N \tilde{f}^i_{t+\theta})_{\theta\geq 0}$ should be easily simulated. In practice, functions $(\tilde{f}^i)_{i=1,\dots,N}$ are taken piecewise constant. We can even think of requiring the condition $\tilde{f}^i_{t+\theta} \geq f^i(X^i_{t+\theta})$ for θ in a small neighborhood of t only and then of choosing $(\tilde{f}^i_{t+\theta})_{\theta\geq 0}$ as a random function depending on the sole past of the system before t, in which case the next event $t + T$ is simulated as the next event in a standard Poisson process.

In fact, the condition $\tilde{f}^i_{t+\theta} \geq f^i(X^i_{t+\theta})$ may be easily verified when we consider the determinisic part of the dynamics (with the sole drift b and without the Brownian motion), as the values of the process X^i are then easily controlled. However, because of the Brownian part in the dynamics, it is impossible to create a piecewise constant approximation function, as the support of a non-degenerate Gaussian random variable is the whole $(-\infty, +\infty)$. However the probability that these values be very large is quite small; as a result, we can choose the approximation function in such a way that $\tilde{f}^i_{t+\theta} \geq f^i(X^i_{t+\theta})$ with very high probability for θ in the neighborhood of t.

For instance, if we consider that, between two spikes t and $t+T$, X^i has the following Ornstein-Ulhenbeck dynamics:

$$dX_\theta^i = -\lambda^i\big(X_\theta^i - a_i\big)d\theta + \sigma^i dW_\theta^i,$$

with X_t^i as initial condition and a_i as attractor, then we have the following explicit expression for the membrane potential:

$$X_\theta^i = a_i + \exp(-\lambda^i(\theta - t))\big(X_t^i - a_i\big) + \sigma^i \int_t^\theta \exp\big(-\lambda^i(\theta - s)\big)dW_s^i, \quad \theta < t + T.$$

In words, the membrane potential follows Gaussian dynamics between two spikes and, using standard confidence intervals for the supremum of Gaussian processes, we can easily cook up the approximation function $\tilde{f}^i$.

Remark. 1. *For very large N, we do not update $\tilde{f}^i$ at each spiking times of one neuron. We only modify the value of $\tilde{f}^{i_0}$ and $\tilde{f}^j$ for neurons j s.t. $J_N^{i_0\to j} \neq 0$.*

2. *The full network is implicitly updated after a fixed time interval: the time is segmented by a coarse time grid of scale Δ, and the $\tilde{f}^i$ are computed within these intervals. At the end of one of these intervals, the maxima $\tilde{f}^i$ are updated. We use this method in order to achieve a trade-off: if Δ is large then the maxima $\tilde{f}^i$ are large which means high computational cost due to the percentage of rejected points whereas if Δ is small then the computational cost is high due to the frequent updates of the full network. The scale Δ is chosen arbitrarily from experience.*

Summary

The algorithm can be found in a *pseudo-code form* in Algorithm 9 and is represented in Figure 2.4. It can be summarized in three steps: generate a spiking time t, select a neuron, apply a rejection sampling condition on the spike. This method is applied as many times as necessary for reaching a certain condition, typically until a final time is reached or a given number of spikes has been found. The rejection method is represented on step (4) in Figure 2.4, but several preliminary steps are necessary.

The step (0) in Figure 2.4 is to find a time interval that should contain the next event. This step is actually important for the speed of the algorithm. Events are exponentially separated, with the sum of the approximation functions as parameter, see step (1) in Figure 2.4:

$$T_{k+1} = T_k + \text{T⊠}, \text{ with T⊠} \sim \mathscr{E}(\sum_i \tilde{f}^i)$$

This step gives the time when the event is occurring. To find the neuron responsible for the event, we use again the sum of the approximation functions. The bigger the value of $\tilde{f}_i$, the higher the probability that the neuron i is the next one to spike.

Once we know which neuron spikes, we update its potential $X^i(T_{next})$ (step (3)). In this regard, it is important to remark that, while the event is categorized as T_{next} in step (1), it is regarded in the end as an event proper to neuron i (2 in the figure); we denote it by $T_{i,n}$.

The value of the potential X^i at time $T_{i,n}$ is needed to compute the value of the rate $f_i(X^i(T_{i,n}))$ and thus to classify the event between false and true spikes (step (4)). The value of a random variable uniformly distributed between 0 and $\tilde{f}_i$ gives the classification: if it is less than $f(X^i(T_{i,n}))$, the spike is a true one; if it is greater, it is a false one. If the spike is a true one, the potential of the spiking neuron is reset to X_R while the potentials of all the postsynaptic (children) neurons are updated to the time of event T_{next}. The values of the synaptic weights $(J^{i\to j})_{j=1,\dots,N}$ are then added to their potentials (step (5)) (so that the rates $(f_j(X^j_{T_{i,n}}))_{j=1,\dots,N}$ may increase). If the spike was a false one, nothing happens.

The algorithms then loops back to step (0) until a given condition (number of spikes reached, time elapsed, etc.) is met.

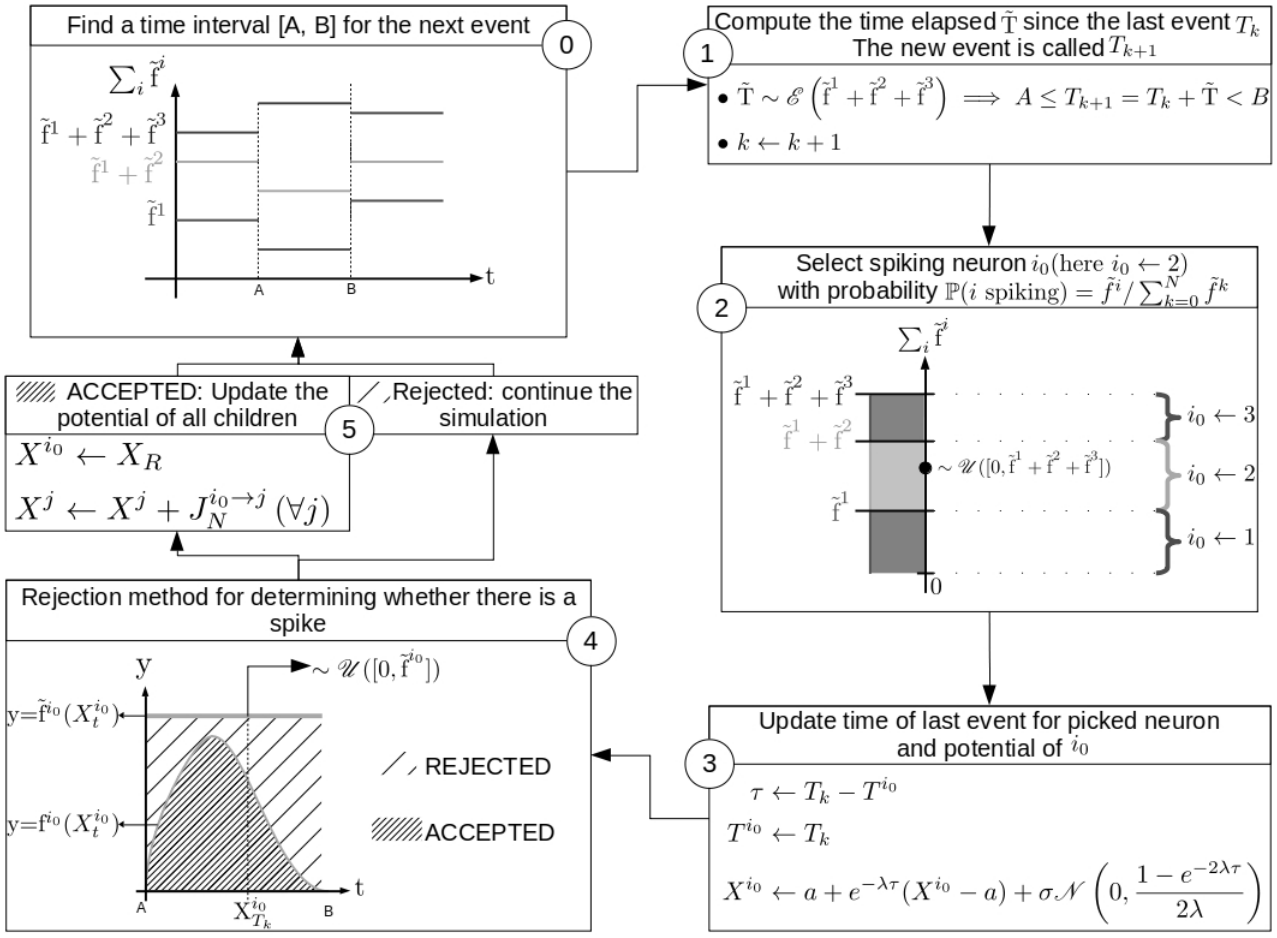


Figure 2.4: Illustration of Algorithm 9 for a system composed of 3 neurons.

The steps (0)→(1)→(2)→(3)→(4)→ (5) are described in the text.

The algorithm (for Ornstein-Ulhenbeck subthreshold dynamics)

Algorithm 9 Simulator

1: N: number of neurons
2: Set $i = k = n = 0$ and $\tau_0 = T_0 = 0$
3: Set $(T^j)_{j\in\{1,\cdots,N\}} = (0)$ and $(X^j)_{j\in\{1,\cdots,N\}} = (0)$
4: Set $i = i + 1$ and generate τ_{i+1}
5: Compute $S = \sum_{n=1}^{N} \tilde{f}^n_{\tau_i}$
6: Set $\tilde{T} \sim \mathscr{E}(S)$ and $\Delta t = \Delta t + \tilde{T}$
7: **if** $\tau_{i-1} + \tilde{T} > \tau_i$ **then**
8: Set $i = i + 1$ and go to 4
9: **end if**
10: Set $k = k + 1$ and put $T_k = T_{k-1} + \tilde{T}$ ▷ System spiking time
11: Put $u \sim \mathscr{U}([0, 1])$
12: $n \leftarrow \mathbf{argmin}_{n\in\{1,\cdots,N\}}\left(\sum_{j=1}^{N} \tilde{f}^j_{\tau_i} * u < \sum_{j=1}^{n} \tilde{f}^j_{\tau_i}\right)$
13: Put $t = T_k - T^n$ and $T^n = T_k$
14: Put $X^n = a^n + e^{-\lambda^n t}(X^n - a^n) + \sigma^n \mathscr{N}(0, (1 - e^{-2\lambda^n t})/(2\lambda^n))$
15: Put $u \sim \mathscr{U}([0, 1])$
16: **if** $\tilde{f}^n_{\tau_i} * u < f(X^n)$ **then**
17: **for all** j **do**1N
18: **if** $j \neq n$ **then**
19: $X^j = a^j + e^{-\lambda^j t}(X^j - a^j) + \sigma^j \mathscr{N}(0, (1 - e^{-2\lambda^j t})/(2\lambda^j))$
20: **end if**
21: $X^j = X^j + J^{n\to j}$
22: **end for**
23: **end if**
24: Put $\tau_i = T_k$ and go to 4

Comparison between clock-driven and event-driven algorithms

We now argue why the event-driven algorithm may be more advantageous than the clock-driven one.

We first recall that the clock-driven method is based upon a time discretization. The state of the system is indeed approximated at discrete times of the form $(T_0 + \ell\Delta)_{\ell\in\mathbb{N}}$, for a given time step Δ. The advantage is pretty clear: The clock-driven method may be easier to implement than the event-driven one. Still, it has a major drawback: the time mesh disregards the own clock of

the system; the former is indeed uniform whilst the latter is deeply heterogeneous since spikes may occur much more frequently at certain periods. In other words, the time mesh chosen in the clock-driven method may not be adapted to the dynamics of the system (we did not explore adaptive time meshes since the event-driven method is automatically adapted). As a result, it may be harder for the clock-driven algorithm to capture the right behavior, leading to precision, stability and complexity issues.

For instance, precision issues may arise because spikes (at least in the simplest form of the clock-driven method) can only occur at the nodes of the time mesh. For sure, this creates a local small error, that may accumulate on the long run. Whilst sophisticated strategies may be implemented to mitigate this effect (for instance the dynamics between two consecutive nodes may be reconstructed *a posteriori* in the form of a bridge), a common (and simple) solution is to take a very small time step Δ. However, the smaller the step Δ the higher the computational cost.

For sure, choosing Δ small has a low interest when the activity of the network is low: When spikes are rare, the computational effort that is required for handling a small Δ looks somewhat disproportionate.

Conversely, Δ has to be chosen very small in order to account for blow-ups. As stated above, this increases the complexity of the method.

These basic observations explain the need for an alternative, event-driven, method. By focusing on the spikes generation (and only computing that), the event-driven approach may indeed reduce the global complexity. Also, times at which the dynamics are simulated are no longer multiples of Δ but can now potentially take any decimal value within the precision range of the machine. The values are then easily precise up to 10^{-15} on modern machines.

2.3.4 Graph management

The storage of the interaction matrix $\boldsymbol{J}_N$ is a strong limitation in our algorithm. To simulate a network with N neurons, we need to know the values of N^2 synaptic weights $\boldsymbol{J}_N^{i\to j}$. In practice, the size of the memory requested to store this matrix is one of the main limitation in our algorithm. For instance, in the classical algorithm used for creating a N-node Erdös-Renyi directed graph, a matrix of size N^2 is created, where each element indicates whether there is a connection between two neurons; typically, this may be an issue when N is greater than 10^6. Indeed, it is impossible to store on a laptop a graph with one million of nodes, or a network with one million of neurons.

The main idea we propose here is to slightly change the classical method for generating the synaptic weights (when given as the realizations of independent and identically distributed variables as it is the case in the Erdös-Renyi graph); in a sentence, our strategy is: "instead of storing N

rows of size N, store only N numbers, namely store for each row one number that encodes the whole row".

We call our new algorithm a *reconstruction* algorithm. Let us rephrase the main idea: instead of storing the full matrix of interaction, we store N values, say $S_0, S_N \dots, S_{N(N-1)}$ (the fact the indices are in the form Ni is explained in Figure 2.5 below). The knowledge of S_{Ni} is sufficient to compute the row $J_N^{i\to\bullet}$ of the synaptic weights from i to all the other neurons. In other words, instead of requesting a memory stack of order $N \times N$, we just request to a smaller memory stack of order N. Of course, the price to pay for this operation is that, each time we need the elements of the row number i, we have to recompute from scratch the whole vector of interactions pointing from i. As we make clear below, this method is numerically efficient if, first, the matrix is static (i.e., the values do not evolve in time), and second, we do not use it too often; in fact, both conditions are fulfilled in our setting.

The whole trick is based on the fact that, from the numerical point of view, the matrix J_N is obtained by a simulation technique, which is in fact deterministic! Consider indeed a Pseudo Random Number Generator (PRNG) in a state S_0. If one uses this PRNG once, a random number r_0 is produced and the state of the PRNG is changed from S_0 to S_1. After N iterations, the PRNG is in a state S_N and N random numbers have been returned. Now if we put the PRNG back to state S_0, and ask again for a number, then r_0 will be returned again and the RNG will be back in state S_1; after N iterations, the PRNG is again in state S_N and the N random numbers that have been returned are the same as the N numbers that had been initially generated. So, going back to the matrix J_N, we can store the states S_{Ni} of the PRNG at the beginning of the simulation of the row $i+1$. Each time we need the values of the weights $J_N^{i_0\to j}$, we just change the state of the PRNG to S_{Ni_0} and generate again (with the PRNG) the *pseudo-random* values $J_N^{i_0\to j}$. Obviously, putting the PRNG in the same state gives the same *pseudo-random* result. We make this clear in Figure 2.5 below.

This method is easily implemented in the case of an Erdös-Renyi graph (which is our benchmark example), see also Algorithm 10. Instead of storing the connections in a matrix, one stores only the first states $(S_{Ni})_{i=0,\cdots,N-1}$; at the end of the day, the memory print is much lower... but the computational cost is higher. To make it clear, this algorithm is not very efficient if one needs to often access the *individual* entries of the matrix, or if one wants a dynamical system where the graph connections are allowed to evolve in time. But in our case the connections are fixed once for all at the beginning of the simulation and the only moment we need them is to update the potential of the children after a spike; in the latter case (and this the key point), what we want is exactly the full row of the matrix.

$$\texttt{Classical method :} \qquad \begin{matrix} S_0 \\ S_N \\ \vdots \\ S_{N(N-1)} \end{matrix} \begin{matrix} \to \\ \to \\ \vdots \\ \to \end{matrix} \begin{pmatrix} J_{11} & J_{12} & \cdots & J_{1N} \\ J_{21} & & & J_{2N} \\ \vdots & & & \vdots \\ J_{N1} & J_{N2} & \cdots & J_{NN} \end{pmatrix} = J$$

$$\texttt{Reconstruction method :} \qquad J = \begin{pmatrix} S_0 \\ S_N \\ \vdots \\ S_{N(N-1)} \end{pmatrix} \begin{matrix} \to J_{11} & J_{12} & \cdots & J_{1N} \\ \to J_{21} & & & J_{2N} \\ \vdots \ \vdots & & & \vdots \\ \to J_{N1} & J_{N2} & \cdots & J_{NN} \end{matrix}$$

Figure 2.5: The two graph management methods summarized here. The → means *"generate the row on the right"*, the $(S_i)_{i\in\{0,\cdots,N(N-1)\}}$ are the state of the PRNG before the generation of the row and $(\cdot)$ is what is kept in memory (graph in the classical method, vector of PRNG's states in the reconstruction method)

Algorithms

Algorithm 10 Generation of a vector of PRNG states

PRNG: Pseudo Random Number Generator
p: probability of connection between two neurons
N: number of neurons in the system
$(V_i)_{i\in\{1,\cdots,N\}}$: vector of PRNG status
for i **do**0N-1
    $V_i \leftarrow S_{i*N}$
    for j **do**1N
        PRNG ▷ State change from $S_{i*N+j-1}$ to S_{i*N+j}
    end for
end for

Remark. *We have to use another random generator number for the simulations of the Brownian motions and the Poisson Processes.*

Algorithm 11 gives a comparison between the two (classical and reconstruction) methods:

Algorithm 11 Comparison of usage between classical method and reconstruction

$(M_{ij})_{(i,j)\in\{1,\cdots,N\}^2}$: Interaction matrix

Neuron i is spiking
▷ Using matrix graph
for j **do**1N
  if $\mathscr{M}_{i,j} = 1$ **then**
    Update potential of neuron j
  end if
end for
▷ Using reconstruction
RNG.SetState(V_i)
for j **do**1N
  if Bernoulli(RNG)= 1 **then**
    Update potential of neuron j
  end if
end for

A natural question is the question of the complexity of such a reconstruction-based method compared to the complexity of the classical one. This is a broader matter that is addressed in the next section.

2.3.5 Complexity: memory and instructions

We remind the user that algorithmic complexity must be read as a function relating the input of an algorithm (more precisely its size) and the number of steps it takes (its time complexity) or the number of storage locations it uses (its space complexity). The asymptotic values of this function (for larger and larger sizes of the input) are noted with a big $\mathscr{O}$ notation.

Of course, the reader must remember that two algorithms with the same order of complexity (or two implementations of the same algorithm) may execute with different amounts of resources because of a multiplicative factor between the two (which will become somehow insignificant asymptotically, but which is not for inputs of small size). Having in mind this limitation, the analysis we provide below focuses on the comparison between the orders of complexity.

	Time complexity at creation	Time complexity during usage	Space complexity
Reconstruction method	$\mathcal{O}(N^2)$	$\mathcal{O}(N)$	$\mathcal{O}(N)$
Interaction matrix	$\mathcal{O}(N^2)$	$\mathcal{O}(1)$	$\mathcal{O}(N^2)$

Table 2.1: Number of operations to create the matrix (col. 1); Number of operations at each call (col. 2); Memory complexity (col. 3).

It must be stressed that the only input that we regard in the analysis of the complexity is the number N of neurons. In particular, all the routines that do not depend explicitly on N do not really appear in the complexity, whilst they could have a non-negligible cost. For instance, PRNG takes time, but it does not show up in the final computation of the complexity.

The first complexity column is given as an example of how the complexity is computed. In the classical algorithm for generating an interaction matrix, for all the parents, we must look at all the other neurons in order to determine whether they are children of the parent. Hence the double loop, leading to N^2. In the case of the reconstruction method presented above, the number of children of a given parent is determined before hand, and in the second loop their index.

It must be also stressed that, in practice, the time complexity at creation of the reconstruction method is lower than the complexity of the interaction method; theoretically, this is not the case as one should take into account the worst case scenario (i.e, the complete graph).

As for the last two columns, the space complexity is obviously smaller for the reconstruction method, making it very relevant for big graphs. Still, the time complexity using the vector of PRNG states is worse than the time complexity of the interaction method, but, as we already commented, this is not such a hindrance in our case.

Acknowledgment. The authors thank Structuring program UCA "Cognitive systems, normality and pathology of the human brain, computational neuroscience", the project "Modélisation Théorique et Computationnelle en Neurosciences et Sciences Cognitives" from University Côte d'Azur, Université Nice Sophia Antipolis, Centre Hospitalier Universitaire de Nice, CNRS and INRIA, for the financial support. This work has been supported by the French government, through the UCAJEDI Investments in the Future project managed by the National Research Agency (ANR) with the reference number ANR-15-IDEX-01. "

References for Chapter 2

1. M. J. Cáceres, J. A. Carrillo, and B. Perthame. "Analysis of nonlinear noisy integrate & fire neuron models: blow-up and steady states". *J. Math. Neurosci.* 1:1, 2011, p. 7.

2. J. A. Carrillo, M. D. M. González, M. P. Gualdani, and M. E. Schonbek. "Classical solutions for a nonlinear Fokker-Planck equation arising in computational neuroscience". *Comm. Partial Differential Equations* 38:3, 2013, pp. 385–409.

3. J. A. Carrillo, B. Perthame, D. Salort, and D. Smets. "Qualitative properties of solutions for the noisy integrate and fire model in computational neuroscience". *Nonlinearity* 28:9, 2015, pp. 3365–3388.

4. E. Catsigeras and P. Guiraud. "Integrate and fire neural networks, piecewise contractive maps and limit cycles". *J. Math. Biol.* 67:3, 2013, pp. 609–655. ISSN: 0303-6812. DOI: 10.1007/s00285-012-0560-7. URL: http://dx.doi.org/10.1007/s00285-012-0560-7.

5. A. De Masi, A. Galves, E. Löcherbach, and E. Presutti. "Hydrodynamic limit for interacting neurons". *J. Stat. Phys.* 158:4, 2015, pp. 866–902. ISSN: 0022-4715. URL: https://doi.org/10.1007/s10955-014-1145-1.

6. F. Delarue, J. Inglis, S. Rubenthaler, and E. Tanré. "Global solvability of a networked integrate-and-fire model of McKean-Vlasov type". *Ann. Appl. Probab.* 25:4, 2015, pp. 2096–2133. DOI: 10.1214/14-AAP1044. URL: https://doi.org/10.1214/14-AAP1044.

7. F. Delarue, J. Inglis, S. Rubenthaler, and E. Tanré. "Particle systems with a singular mean-field self-excitation. Application to neuronal networks". *Stochastic Process. Appl.* 125:6, 2015, pp. 2451–2492.

8. S. Delattre, G. Giacomin, and E. Luçon. "A note on dynamical models on random graphs and Fokker-Planck equations". *J. Stat. Phys.* 165:4, 2016, pp. 785–798.

9. P. Erdős and A. Rényi. "On random graphs. I". *Publ. Math. Debrecen* 6, 1959, pp. 290–297.

10. P. Erdős and A. Rényi. "On the evolution of random graphs". *Magyar Tud. Akad. Mat. Kutató Int. Közl.* 5, 1960, pp. 17–61.

11. N. Fournier and E. Löcherbach. "On a toy model of interacting neurons". *Ann. Inst. Henri Poincaré Probab. Stat.* 52:4, 2016, pp. 1844–1876. ISSN: 0246-0203. URL: https://doi.org/10.1214/15-AIHP701.

12. E. Gobet. "Weak approximation of killed diffusion using Euler schemes". *Stochastic Process. Appl.* 87:2, 2000, pp. 167–197. ISSN: 0304-4149. URL: https://doi.org/10.1016/S0304-4149(99)00109-X.

13. A. J. Graham and D. A. Pike. "A note on thresholds and connectivity in random directed graphs". *Atl. Electron. J. Math.* 3:1, 2008, pp. 1–5.

14. B. Hambly, S. Ledger, and A. Søjmark. "A McKean–Vlasov equation with positive feedback and blow-ups". *ArXiv e-prints*, 2018. arXiv: 1801.07703.

15. A. L. Hodgkin and A. F. Huxley. "A quantitative description of membrane current and its application to conduction and excitation in nerve". *J. Physiol.* 117:4, 1952, pp. 500–544.

16. L. Lapicque. "Recherches quantitatives sur l'excitation électrique des nerfs traitée comme une polarisation". *J. Physiol. Pathol. Gen.* 9, 1907, pp. 620–635.

17. T. J. Lewis and J. Rinzel. "Dynamics of spiking neurons connected by both inhibitory and electrical coupling". *J. Comput. Neurosci.* 14:3, 2003, pp. 283–309.

18. S. Nadtochiy and M. Shkolnikov. "Mean field systems on networks, with singular interaction through hitting times". *ArXiv e-prints*, 2018. arXiv: 1807.02015.

19. S. Nadtochiy and M. Shkolnikov. "Particle systems with singular interaction through hitting times: application in systemic risk modeling". *ArXiv e-prints*, 2017. arXiv: 1705.00691.

20. Y. Ogata. "On Lewis' simulation method for point processes". *IEEE Transaction on Information Theory* 27:1, 1981, pp. 23–31.

21. S. Ostojic, N. Brunel, and V. Hakim. "Synchronization properties of networks of electrically coupled neurons in the presence of noise and heterogeneities". *J. Comput. Neurosci.* 26:3, 2009, pp. 369–392.

22. I. Palásti. "On the strong connectedness of directed random graphs". *Studia Sci. Math. Hungar* 1, 1966, pp. 205–214. ISSN: 0081-6906.

23. A.-S. Sznitman. "Topics in propagation of chaos". In: *École d'Été de Probabilités de Saint-Flour XIX—1989*. Vol. 1464. Lecture Notes in Math. Springer, Berlin, 1991, pp. 165–251.

3 Article 2: Efficient simulation of sparse graphs of point processes

Written by C. Mascart, A. Muzy, P. Reynaud-Bouret

A new algorithm for multivariate point process simulation, and in particular Hawkes point process with piecewise constant intensities, is exposed in this second article. The algorithm is derived from the algorithm proposed by Ogata in [106] and developed in section 1.2.8. The theoretical complexity, depending on the number of processes, of the new algorithm is compared with the theoretical complexity of the classical algorithm developed by Ogata, adapted here for piecewise constant functions to keep the comparison fair. The execution time of the two algorithms are also compared using different topologies, and a discrete sampling of the parameter space. Finally goodness-of-fit tests are applied to the output of both algorithms to verify the quality of the simulation.

My contributions to this article consists of successive designs and implementation of the algorithm, as well as running the simulations, collecting the results and executing the goodness-of-fit tests.

Status: Submitted and accepted in TOMACS

3.1 Abstract

We derive new discrete event simulation algorithms for marked time point processes. The main idea is to couple a special structure, namely the associated local independence graph, as defined by Didelez [13], with the activity tracking algorithm [23] for achieving high performance asynchronous simulations. With respect to classical algorithm, this allows reducing drastically the computational complexity, especially when the graph is sparse.

3.2 Introduction

Point processes in time are stochastic objects that model efficiently event occurrences. The variety of applications is huge: from medical data applications (time of death or illnesses) to social sciences (dates of crimes, weddings, etc), from seismology (earthquake occurrences) to micro-finance (actions of selling or buying a certain assets), from genomics (gene positions on the DNA strand) to reliability analysis (breakdowns of complex systems) (see e.g. [1, 9, 13, 25, 30, 32]).

Most of the time, point processes are multivariate, in the sense that either several processes are considered at the same time, or in the sense that one process regroups together all the events of the different processes and marks them by their type. A typical example consists in considering either two processes, one counting the wedding events of a given person and one counting the birth dates of children of the same person. One can see this as a marked process which regroups all the possible dates of birth or weddings independently and on each event one marks it by its type, here wedding or birth.

In the sequel, we denote the individual process N_j, the set of all events corresponding to type j, for $j = 1, ..., M$ and the joint process $N = N_1 \cup .. \cup N_m$. In this multivariate or marked case, the individual processes are usually globally dependent, the apparition of one event or point on a given type influencing the apparition of other points for the other types and the simulation of the whole system cannot be easily parallelized.

This is especially true in neuroscience [29]. Let us detail a bit more this set up which is a benchmark example here. Neurons are excitable electric cells that are linked together inside a huge network (10^{11} for humans [3], 10^8 for rats [19], 10^6 for coackroaches), each cell receives inputs from approximately 10^3 to 10^4 presynaptic (upstream) neurons [27]. Depending on its excitation, the neuron might then produce an action potential also called spike, information which is propagated to postsynaptic (downstream) neurons.

From a stochastic point of view, one might then see the spike trains emitted by a given neuron as an individual point process which in fact is embedded in a multivariate point process with M, the total number of neurons as the total number of types. The size of the network requires then

very well adapted simulation schemes that may use the relative sparseness of the network with respect to the global size of the network.

To do so, we use the mathematical notion of local independence graph for marked point processes due to Didelez [13], which is detailed in Section 3.5 and which informally corresponds to the real neuronal network. In this sense, *in the sequel we call marks, processes and type the nodes of the graph*. The only strong assumption that is used is the *time asynchrony hypothesis*, (i.e. points or events of different mark or types, meanings points or events appearing in different nodes, cannot occur at the exact same time) together with the fact that all processes have a *conditional intensity* [6].

Simulation of point processes has a long history that dates back to Doob in the 40's [14] for Markov processes. In the 70's, Gillespie [16] popularized the method for a particular application: chemical reactions. At the same time, Lewis and Shedler [20] proposed a thinning algorithm for simulating univariate inhomogeneous Poisson processes (this can also be viewed as a rejection method). Few years later, Ogata [26] produced a hybrid algorithm able to simulate multivariate point processes in the general case even if they are not markovian, including both a choice of the next point by thinning and a choice of the node to activate thanks to Gillespie principle. This method is still up to now the benchmark for simulating such processes, and is for instance used in recent packages such as `ppstat` in `R` (2012). It has been rediscovered many times in various cases, most of the time as a Generalized Gillespie method (see for instance [2]).

When the number of types or nodes is huge, this method can quickly become inefficient in terms of computational times. Many people have found shortcuts, especially in Markovian settings. For instance, Peters and de With [28] proposed a new simulation scheme exploiting a network of interaction for particular physical applications. In [4], the authors reformulated this algorithm in a more mathematical way for a particular case of Piecewise Deterministic Markov Processes. People have even exploited very particular structures such as Hawkes processes, with exponential interactions (special case which leads to Markovian intensities) [12], to be able to simulate huge networks, as in the `Python` package `tick` (2017).

In the mean time, the technique of discrete event simulation first appeared in the mid-1950s [31] and was used to simulate the components (machines) of a system changing state only at discrete "events". This technique has then been formalised in the mid-1970s [33]. Discrete event modelling and simulation seem very close to point process models (dealing with events, directed graphs, continuous time, etc.). Against all expectations, as far as we know, there is no direct use of any discrete event simulation algorithm for point processes. Maybe, the sophistication of these algorithms being of the same order than the mathematical technicality of point processes, prevented any direct application. Besides, the continuous nature of the conditional intensity associated to a point process with respect to the discreteness of event-based simulations could make appear the two

domains as separated whereas discrete event theory is a computational specification of mathematical (continuous) systems theory [22] integrating more and more formally stochastic simulation concepts [34]. We hope to show here that both domains can take advantage from each other, by introducing new discrete-events models that act as generators of point processes. Especially, whereas discrete event simulation algorithms have been developed considering independently the components (nodes) of a system, a new algorithm for activity tracking simulation [23] have been proposed to track activity (events from active nodes to children). The activity tracking algorithm is used here and proved to be the right tool for both simplifying usual discrete event algorithms (which are difficult to relate to usual point process algorithms) and efficiently simulating point processes.

Our aim is to derive a new simulation algorithm, which generalises the algorithm of [4] to general multivariate point processes that are not necessarily Markovian, by exploiting the underlying network between the types, which is here a local independence graph. In Section 3.3, the main mathematical background and notations are provided and the classical multivariate algorithm due to Ogata [26] is explained. A simplified version in discrete event terms and called *full scan algorithm* is proposed. In Section 3.4, discrete event data structure and operations specific to point processes are designed. In Section 3.5, after recalling the notion of local independence graph [13], a new *local graph algorithm* is presented. In Section 3.6, we evaluate the computational complexities of both algorithms on Hawkes processes with piecewise constant interactions, which model easily neuronal spike trains [29]. We show that in this case, for sparse graphs, new local graph algorithm clearly outperforms the classical Ogata's algorithm in its discrete event version.

3.3 Set-up

3.3.1 Mathematical framework

A (univariate) point process N in $\mathbb{R}_+$ is a random countable set of points of $\mathbb{R}_+$. For any subset A of $\mathbb{R}_+$, $N(A)$ is the number of points that lie in A.

As real random variables might be defined by their density with respect to Lebesgue measure, if it exists, a point process is characterised by its conditional intensity with respect to a given filtration or history $(\mathcal{F}_t, t \geq 0)$. For the mathematical details, we refer the reader to [6]. Informally, the filtration or history at time $t-$, $\mathcal{F}_{t-}$, contains all the information that is needed to simulate the next point of N, when one is just before time t. It usually includes as generators, all the points

$T \in N$ such that $T < t$ in particular. The (conditional) intensity of the point process N is then informally defined [6] by

$$\lambda(t) = \lim_{dt \to 0} \frac{1}{dt} \mathbb{P}\Big(\text{there is a point of } N \text{ in } [t, t+dt] \Big| \mathscr{F}_{t-} \Big),$$

for infinitesimal dt, where $\mathbb{P}\Big(\text{there is a point of } N \text{ in } [t, t+dt] \Big| \mathscr{F}_{t-} \Big)$ is the probability that a point appears in the interval $[t, t+dt]$ given what happened strictly before t in the history. This is a random process which, at time t, may depend in particular on all the past occurrences of the process itself, that is the $T < t$.

A multivariate point process can be seen as a collection of M different point processes N_j. With the *time asynchrony hypothesis,* one can also consider equivalently the univariate joint point process $N = N_1 \cup ... \cup N_M$ and say that for each point t of N there is one and only one subprocess j such that $t \in N_j$. This j is then called the mark of point t, or the node associated to t.

We are given the set of intensities of each of the N_j, $t \mapsto \lambda_j(t)$, with respect to a common filtration $(\mathscr{F}_t, t \geq 0)$. Note that $\mathscr{F}_{t-}$ includes as generators, all the points $T \in N$, the joint process such that $T < t$ as well as their respective marks.

Examples Let us give just few basic examples

- **Homogeneous Poisson processes** with rates $(\nu_i)_{i=1,...,M}$. In this case, all λ_i are constant and not even random and for all i,

  $$\lambda_i(t) = \nu_i.$$

  We see in this expression, that the intensities do not depend neither on time, nor on the previous occurrences. This is why one often refers to such dynamics as "memoryless". Since these processes do not interact, one can of course simulate each N_i in parallel if need be. In this case, for each of them, it is sufficient to simulate the time elapsed until the next point, by an exponential variable of parameter ν_i, independently from anything else. To unify frameworks, this exponential variable might also be seen as $-\log(U)/\nu_i$, with U a uniform variable on $[0, 1]$.

- **Inhomogeneous Poisson processes** with time-dependent rates $(f_i)_{i=1,...,M}$. In this case, the λ_i's are not necessarily constant and but they are still non random and for all i,

  $$\lambda_i(t) = f_i(t).$$

Once again parallelization is possible, and for each individual process N_i and given point t_k^i, one finds the next point t_{k+1}^i by solving

$$\int_{t_k^i}^{t_{k+1}^i} f_j(s)\ \mathrm{d}s = -\log(U)$$

- **Linear multivariate Hawkes process** with spontaneous parameter $(\nu_j)_{j=1,\dots,M}$ and non negative interaction functions $(h_{j\to i})_{i,j=1,\dots,M}$ on $\mathbb{R}_+$. This process has intensity

$$\lambda_i(t) = \nu_i + \sum_{j=1}^{M} \sum_{T\in N_j, T<t} h_{j\to i}(t-T). \tag{3.1}$$

This process is used for many excitatory systems, especially the ones modelling the spiking activity of neurons [29]. It can be interpreted in this sense, informally: to a homogeneous Poisson process of rate ν_i, which models the spontaneous activity of the neuron i, one adds extra-points coming from the interactions. Typically a point T of mark (neuron) j adds a term $h_{j\to i}(\delta)$, after delay δ to the intensity of N_i making the apparition of a new point at time $t = T + \delta$ more likely. In this sense there is an excitation of j on i. Here we see a prototypical example of global dependence between the marks. Each new point for each mark depends on all the points that have appeared before, with all the possible marks, preventing a brute force parallelization of the simulation. Except when the $h_{j\to i}$'s are exponentially decreasing [12], this process is clearly not Markovian. It is for this kind of general process that one needs efficient simulation algorithms.

3.3.2 Simulation of univariate processes

The time-rescaling theorem (see [8] or [6] for more mathematical insight) states that if a point process N has a conditional intensity $\lambda(t)$, and if

$$\forall t, \Lambda(t) = \int_0^t \lambda(s)\ \mathrm{d}s,$$

then $\mathcal{N} = \{\Lambda(T), T \in N\}$ is a Poisson process of rate 1. This is why, even for general point processes, it is always possible to find, by iteration the next point of N by solving recursively, for all $k \in \mathbb{N}^*$ the set of positive natural numbers,

$$\int_{t_k}^{t_{k+1}} \lambda(s)\ \mathrm{d}s = -\log(U) \tag{3.2}$$

initializing the method with $t_0 = 0$.

Of course, to be able to mathematically solve this easily, one needs to be able at time t_k to compute $\lambda(t)$ on $(t_k, +\infty)$ if no other point occurs. This in particular happens if the filtration $\mathcal{F}_t$ is reduced to the filtration generated by the points themselves and this is what we will assume here. Of course all algorithms discussed here can easily be adapted to richer filtrations, as long as the computation of $\lambda(t)$ on $(t_k, +\infty)$ can be carried out (if no other point occurs).

In this situation, two cases might happen, each of them leading to a different algorithm:

Transformation method: The function $\lambda(t)$ on $(t_k, +\infty)$ (and if no other point occurs) has an easily computable primitive function with inverse $\Lambda^{-1}(t)$. Then (3.2) reduces to

$$t_{k+1} = \Lambda^{-1}(-\log(U) + \Lambda(t_k)).$$

Thinning method: It applies if the previous computation is not possible or easy but one can still compute $\lambda^*(t) \geq \lambda(t)$ such that $\lambda^*(t)$ has all the desired properties of the transformation method (typically $\lambda^*(t)$ is constant, with constant that might depend on the t_ℓ for $\ell \leq k$). Then the algorithm does as follows to compute a possible next point (cf. Algorithm 12). If thinning for Poisson processes is due to [20], it has been generalized to general processes by Ogata [26]. One can find a complete proof in [10].

Algorithm 12 Thinning algorithm

1: initialize $t_0^* \leftarrow t_k$
2: **repeat**
3: Generate next point t^* after t_0^* of a point process with intensity function λ^* by the Transformation method.
4: Generate $U \sim \mathcal{U}[0, 1]$
5: **if** $U > \lambda(t^*)/\lambda^*(t^*)$ **then** # Rejection
6: $t_0^* \leftarrow t^*$
7: **end if**
8: **until** $U \leq \lambda(t^*)/\lambda^*(t^*)$
9: **return** $t_{k+1} \leftarrow t^*$

3.3.3 Discrete event version of classical multivariate algorithm for point processes

To simulate multivariate processes, Ogata [26] made an hybridation between two different notions : the thinning algorithm presented above and the attribution of marks. Indeed, since all processes N_j are communicating with each other, Ogata's idea for the attribution of marks is to generate the next point of the aggregated process N and, then, to decide for its mark.

In the present article, we choose to discard the thinning part, which can be added to all the algorithms that we derive here. The attribution part of Ogata's multivariate algorithm [26], is referred in the sequel as *full scan*, because the intensities of all nodes of the graph need to be scanned and updated at each time stamp t_k. Once the next point of N is decided, the basic idea for the attribution of marks is to attribute them at random, the distribution taking into account the relative value of the intensity of each subprocess. The main steps of this algorithm are presented in Figure 3.1 for a visual representation of the method. More details about the algorithm steps are provided through the Hawkes application in Section 3.6.

Algorithm 13 Full scan multivariate algorithm modified from [26]

1: $t_0 \leftarrow 0$
2: **while** $t_k < T$ **do**
Step a/ 3: **Compute intensity sums** $\sum_{j=1}^{i} \lambda_j(t) = \overline{\lambda}_i(t)$, for $i \in \{1, ..., M\}$ on $t \in (t_k, +\infty)$
Step b/ 4: **Get** by simulation t_{k+1} as the **next point** of a univariate point process of intensity $\overline{\lambda}_M(t)$
Step c/ 5: **Select the unique possible node** i_{k+1}, such that $\frac{\overline{\lambda}_{i_{k+1}-1}(t_{k+1})}{\overline{\lambda}_M(t_{k+1})} < V \leq \frac{\overline{\lambda}_{i_{k+1}}(t_{k+1})}{\overline{\lambda}_M(t_{k+1})}$, with $V \sim \mathcal{U}[0,1]$ and $\overline{\lambda}_0 = 0$
Step d/ 6: **Update intensities** λ_j on $(t_{k+1}, +\infty), \forall j \in \{1, ..., M\}$
7: $k \leftarrow k + 1$
8: **end while**
9: **return** points $(t_1, ..., t_{k-1})$ and associated nodes $(i_1, ..., i_{k-1})$

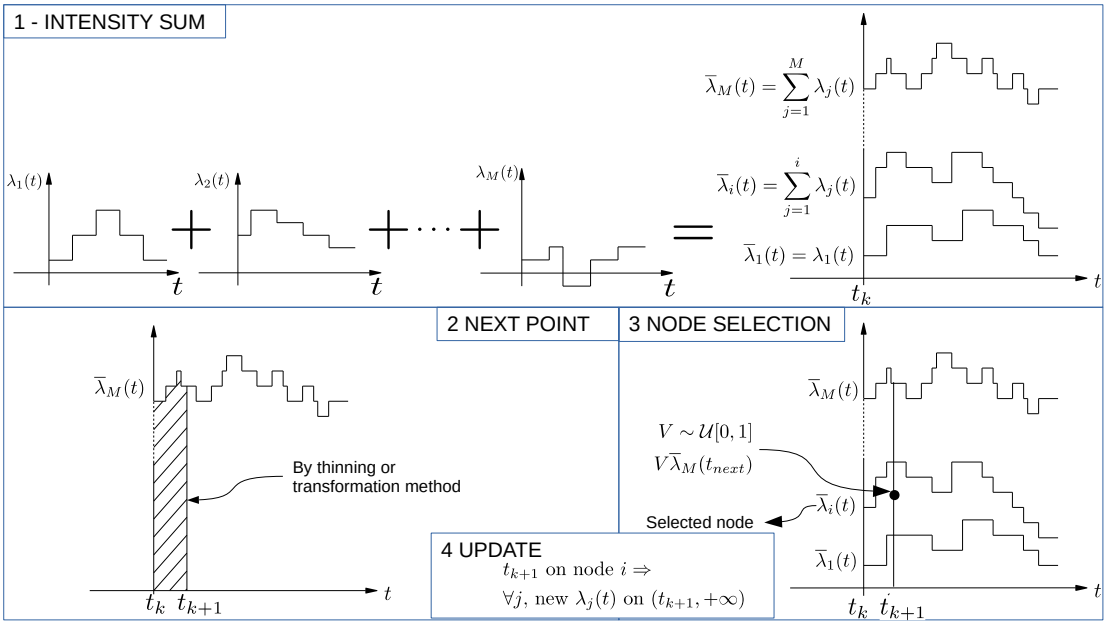


Figure 3.1: Steps of the full scan algorithm for point processes. The intensities are piecewise constant (cf. Section 3.6).

Original Ogata's algorithm uses thinning at step 4 of Algorithm 13. However the complexity of a thinning step is difficult to evaluate because it depends on both the complexity of the upper-bounding function λ^* and how far this function is from λ, which influences how much time the thinning algorithm rejects. Therefore for a clear evaluation of the complexity, we focused on simulations where the transformation method is doable, typically when the intensities are piecewise constant.

3.4 Specific discrete event data structures and operations

Before introducing our new algorithm, we present a particular structure, which is very important for discrete events algorithm : the scheduler.

A *scheduler* Q is a data structure, which can be represented as an ordered set of *events*, and which is provided with a set of operations for ensuring the correct order of the events. The events are noted $ev_i = (t_i, v_i)$, where t_i is the *event time* and v_i is the *event value*. The events in the scheduler are increasingly ordered in time, i.e., $ev_i, ev_j \in Q, ev_i < ev_j \iff t_i < t_j$. The length of the scheduler is noted $|Q|$.

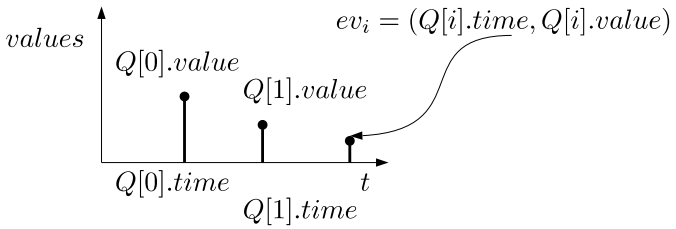


Figure 3.2: Example of events in scheduler Q. For an event ev_i in the scheduler, the *value* is accessed by $Q[i].value$ and the *time* is accessed by $Q[i].time$, with $i = 0$ the first event index.

In both full scan and local graph algorithms schedulers are used. They are usually implemented using one of the many kinds of self-balancing binary tree. The choice of structure (AVL, red-black, etc.) depends on corresponding operation complexity. Here we choose the red-black self balancing tree, which exhibits the best performance with respect to other usual self-balancing trees [18].

To make a scheduler, the red-black tree is equipped with a set of classical (**insert**, **remove**, **upper bound** and **lower bound**) and non-classical operations. All these operations, except stated otherwise, have time complexity bounded by the logarithm $\log_2$ of the number of elements in the set. In addition it is supposed that any element of the scheduler can be accessed with a constant time. This may not be the case depending on the language used for the implementation. However, it is true for C++ language, which was used in our implementation.

3.4.1 Basic *scheduler operations*

We describe and illustrate the set of operations completing the red-black tree to form the scheduler data structure needed for our algorithms. Any operation on a scheduler Q directly modifies it. Also, to simplify the operations, it is supposed that all events stored in a scheduler Q are unique.

Access element operation $(t^i, v^i)_{i\in\{0,\dots,|Q|-1\}}$ is the list of events stored in scheduler Q and sorted by ascending values of t. Then operation $Q[i]$ returns event (t^i, v^i) with constant time complexity $\mathcal{O}(1)$.

Insert operation of an event $(t, v) \in \mathbb{R}_+ \times \mathbb{R}$ (cf. Figure 3.3): $Q \oplus (t, v)$ inserts the event in the tree. This operation has a total maximum complexity of $\mathcal{O}(log_2(|Q|))$, to find the place of the event and rebalance the tree.

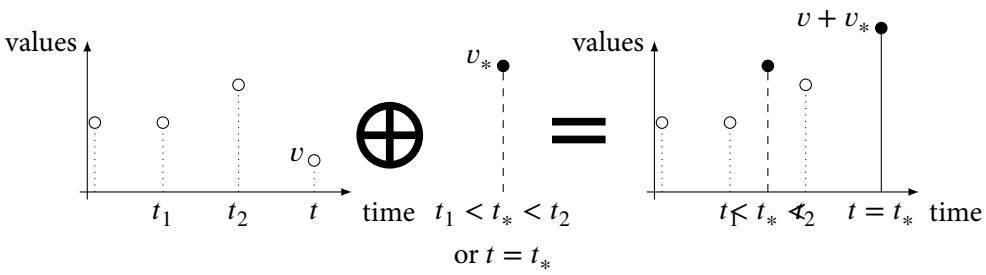


Figure 3.3: A graphical example of the insertion of a new event (t_*, v_*) inside a scheduler. There are two cases: if the event time is unmatched in the set of event times already present in the scheduler (case $t_1 < t_* < t_2$), the event is just inserted in the right place; otherwise (case $t = t_*$) the event in the scheduler with the same time has its value increased by the value v_* of the new event.

Remove operation of an event (cf. Figure 3.4): $Q \ominus (t, v)$, removes the event (t, v) from the tree. This operation has a maximum complexity of $\mathcal{O}\big(\log_2(|Q|)\big)$, to find the place of the event and rebalance the tree. If the index i of the element is known, the remove operation can be executed in constant time.

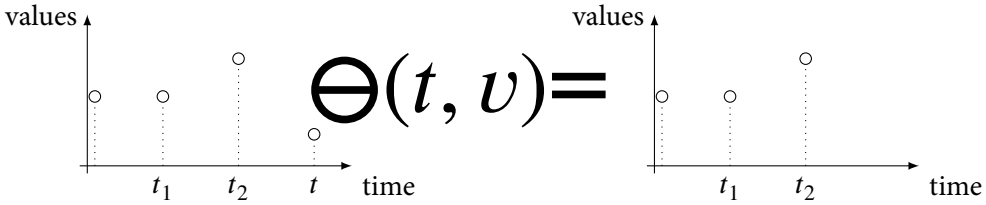


Figure 3.4: A graphical example of the removal of an event given its time t.

Remove first operation Q^* over the scheduler Q, which removes the first event $Q[0] = (t^0, v^0)$ from the scheduler, the second event becoming the first. Operation $\cdot^*$ has complexity $\mathcal{O}(1)$.

Prune operation of the scheduler Q^t (cf. Figure 3.5) removes all events (t^*, v^*) from $|Q|$ whose time value $t^* \leq t$. The operation Q^t has complexity $\mathcal{O}\big(log_2(|Q|) + |\text{number of events until time } t|\big)$.

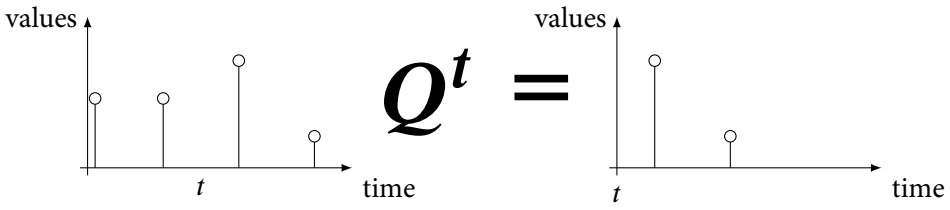


Figure 3.5: A graphical example of the prune operation: here all the points with a time less or equal to t (here the first two points of this scheduler) are removed. The time complexity of the operation is $\mathcal{O}(\log_2(4) + 2)$ (scheduler of size $|Q| = 4$ and 2 events removed).

Upper and lower bound operations (cf. Figure 3.6): Upper bound operation: $\lceil t \rceil_Q$ and lower bound operation $\lfloor t \rfloor_Q$, return the smaller event (t^*, v^*) verifying respectively $t^* > t$ and $t^* \geq t$. In both cases the time complexity of the operation is $\mathcal{O}(log_2(|Q|))$.

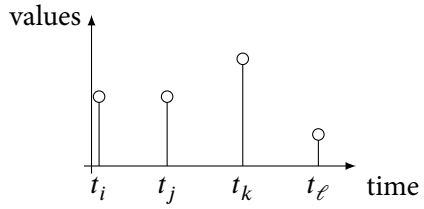


Figure 3.6: Upper/lower bound operations on an example of scheduler Q. For $t_k < t < t_\ell$ the *upper bound operation* consists of $\lceil t \rceil_Q = \ell$. The *lower bound operation* consists of $\lfloor t \rfloor_Q = k$.

3.4.2 Using a scheduler to represent piecewise constant functions

Let $h : \mathbb{R}^* \to \mathbb{R}, t \mapsto h(t)$ be a piecewise constant function with finite support $S \subseteq \mathbb{R}^*$. A scheduler can encode piecewise constant functions on S. There are two possible methods to achieve this:

1. Let each discontinuity $(t, h(t))$ be its own event in the scheduler Q representing the function h (see top part of Figure 3.7).

2. Represent the discontinuity not as a new value $(t, h(t))$ but as an increment, that is a difference between the new value of h after the discontinuity and the value of h just before: $(t, h(t) - h(t\_))$ (see bottom part of Figure 3.7).

The first representation is more straightforward, and is very efficient when the function is stored only for lookups: the function can then be stored in an array and a dichotomic search algorithm be used to locate any event in logarithmic time.

However, if discontinuities must be introduced or removed, for instance by adding another piecewise constant function to h, then some of the events stored in the scheduler may need to be changed or moved in the scheduler, thus creating an undesirable overhead. For instance in the first line of figure 3.7, a piecewise function h (first column) is represented and encoded (second column) with the straightforward method by a scheduler. Last column shows the new encoding representing a sum. The discontinuity at t_{k+2} had to be modified because of the introduction of discontinuities at t_{k+3} and t_{k+4}. For a function with many more discontinuities, or when summing two piecewise functions of similar size, the operation would necessitate to modify a large part of the already represented points.

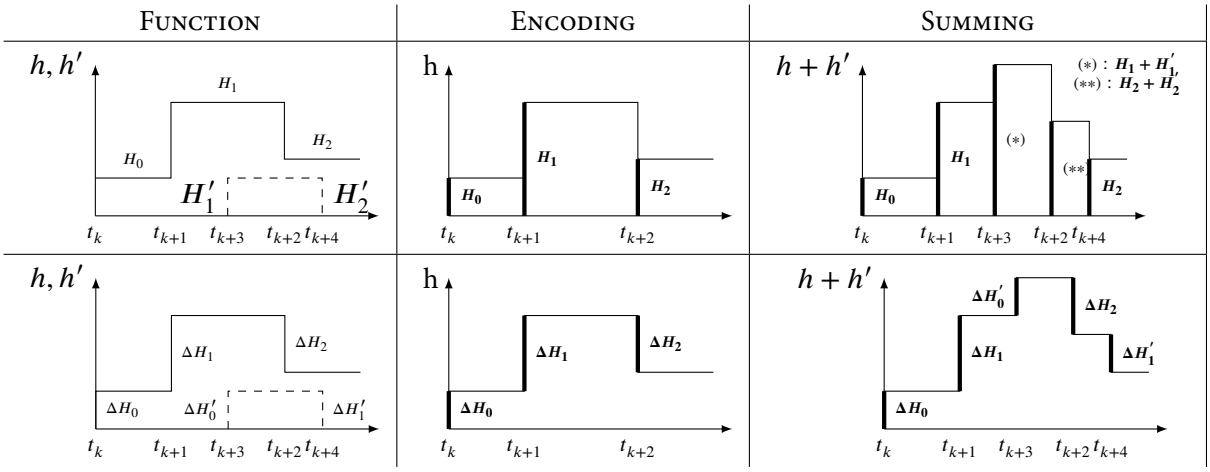


Figure 3.7: Example of a straightforward encoding of a piecewise constant function (first line) and an optimised one (second line). We use the notation $H_n = h(t_{k+n}), n \in \mathbb{N}$ for concision. The FUNCTION column represents the piecewise constant function h to encode (plain lines), as well as another function h' (dashed lines, see column SUMMING). Column ENCODING represents in bold the events $(t_n, h(t_n))$ encoded by a scheduler representing the function h. Column SUMMING represents in bold the values encoded by a scheduler now representing the function $h + h'$.

The solution we proposed is to store not the values $h(t)$ of the function h at a time t where a discontinuity happens, but the variation $\Delta h_t = h(t) - h(t\_)$ of the value of h during the discontinuity. An example is represented on the second line of Figure 3.7.

Let us now present two different operations for piecewise contant functions with this encoding:

Piecewise sum, which is in fact a union operation over two schedulers Q, Q' encoding two piecewise constant functions (cf. Figure 3.8): The operation $Q \cup Q'$ has complexity $\mathcal{O}(min(|Q|, |Q'|)log_2(max(|Q|, |Q'|)))$, since the smallest scheduler is inserted into the largest scheduler.

Piecewise prune operation of the scheduler Q^t_{pcw} (cf. Figure 3.9): removes from scheduler Q all the events (t^*, v^*) with $t^* \leq t$, while accumulating all the values v^* up to time t. At the end of the operation the scheduler begins with an event of the form $(t, \sum v^*)$. The operation Q^t_{pcw} has complexity $\mathcal{O}(log_2(|Q|) + |\text{number of events until time } t|)$.

Shift operation of the scheduler $Q_{\rightarrow_t}$ (cf. Figure 3.10): shifts all points by t. The operation $Q_{\rightarrow_t}$ has complexity $\mathcal{O}(|Q|)$.

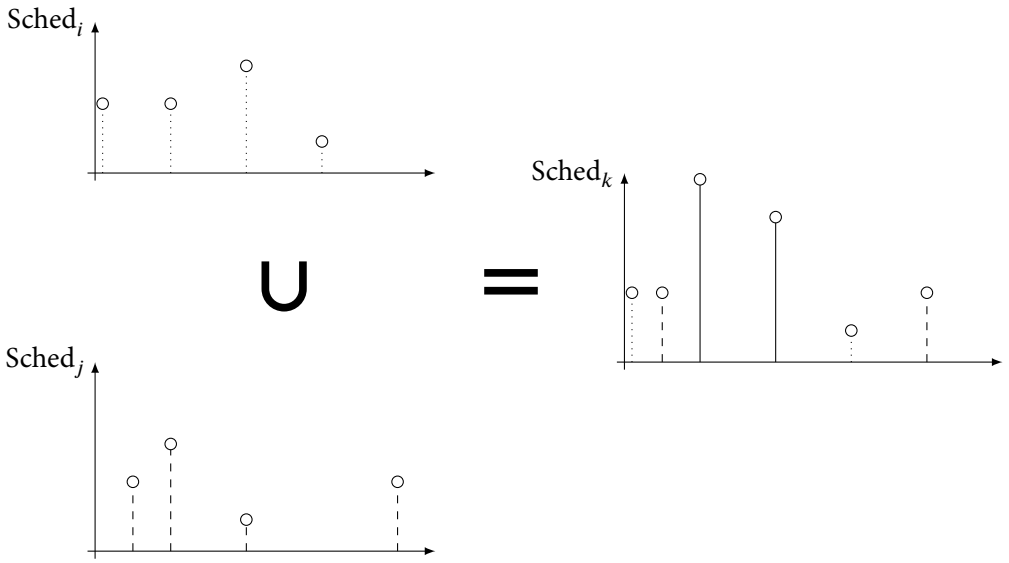


Figure 3.8: A graphical example of the union of two schedulers. The events of Sched$_i$ are represented as dotted, while the events of Sched$_j$ are dashed. In the merged scheduler Sched$_k$, the values of events at the same time in both schedulers i and j are summed and the resulting event is represented with a continuous line.

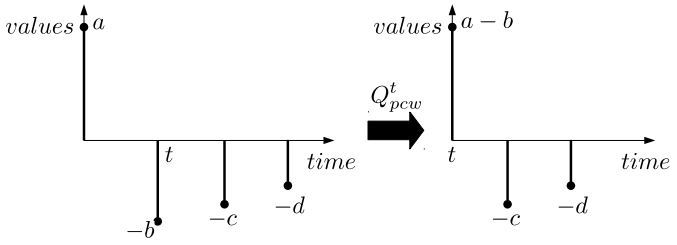


Figure 3.9: Example of piecewise prune operation Q^t_{pcw}.

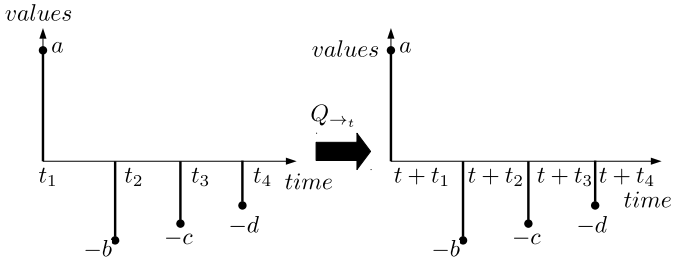


Figure 3.10: The shift operation $Q_{\rightarrow_t}$

3.5 Local graph algorithm for point processes

3.5.1 Local independence graph

Local independence graphs are fully presented in a sound mathematical form in [13]. For a given multivariate point process $N_j, j = 1, ..., M$, the corresponding local independence graph is a directed graph on the nodes $j = 1, ..., M$ (see for instance Figure 3.11). We assume for sake of simplicity that the filtration is reduced to the internal history, that is $\mathcal{F}_t$ is generated only by the $T < t$ in $N = N_1 \cup ... \cup N_M$ and their associated mark or node.

To explain more fully what a local independence graph means, we need to define rougher filtration. For a subset $I \subset \{1, ..., M\}$, $\mathcal{F}_t^I$ is the filtration generated by the $T < t$ in $\cup_{i \in I} N_i$ and their associated node.

In a local independence graph, the absence of edge $j \to i$ means that the apparition of a point at time t in N_i is independent from $\mathcal{F}_{t-}^{\{j\}}$ conditionally to $\mathcal{F}_{t-}^{\{j\}^c}$, where $\{j\}^c = \{1, ..., M\} \setminus \{j\}$.

So this means that for every time t, the intensity $\lambda_i(t)$ of N_i does not depend directly on the positions of the points of N_j strictly before t.

This extends directly to the notion of parents and children in the graph. For a given node i, one defines

$$pa(i) = \{j, j \to i \text{ is in the graph}\} \text{ and } ch(i) = \{j, i \to j \text{ is in the graph}\}.$$

Therefore it means that the intensity $\lambda_i(t)$ at time t of N_i in fact only depends on the points of N_j for $j \in pa(i)$ strictly before t.

Conversely, a point on N_i directly impacts the occurrence of points for N_j for $j \in ch(i)$. Note that in any case, it also impacts the next point of N_i because even for a Poisson process without memory one needs by the transformation method to know t_k for finding t_{k+1}. However, it will not have any direct impact on the future points of N_j for $j \notin ch(i) \cup \{i\}$.

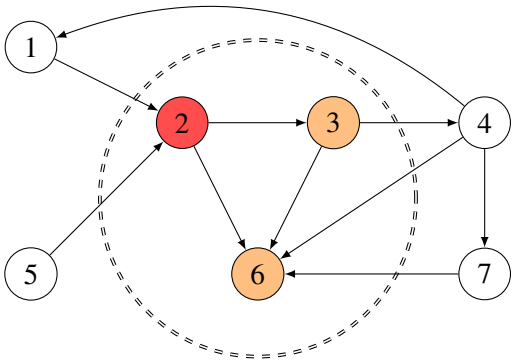


Figure 3.11: Example of local independence graph. With this graph, $ch(2) = \{3, 6\}$ and $pa(2) = \{1, 5\}$. As indicated by the difference of colour, a point with mark 2 shall impact the point generation only for $\{2\} \cup \{3, 6\}$.

For instance, for linear Hawkes process (cf. Equation 3.1), $ch(i) = \{j / h_{i \to j} \neq 0\}$.

3.5.2 Local-graph algorithm

The children of each node are stored in a simple one dimensional array, whose indexes are the node indexes, and whose cells contain vectors of the children indexes. So accessing a node simply costs $\mathcal{O}(1)$.

Because of the interpretation of $I = ch(i) \cup \{i\}$ of a given node i given above in the local independence graph, it means that in fact, after having simulated t_k with mark/node i_k in the joint process, we know that only the next points of N_j for $j \in I$ have to be modified.

At simulation level, discrete events are used to track activity nodes associated to selected points (time stamps) to their children. Discrete events are stored into a scheduler Q of events $ev_i = (t^i_{next}, i)$, where t^i_{next} is the *possible next point* associated to node i.

The local graph algorithm for point processes is described in Algorithm 14. A visual representation is presented in Figure 3.12.

Let us detail a bit more each step.

- At Step a/, we decide for the next possible point of each of the subprocesses N_i by either using the transformation method or the thinning algorithm (Algorithm 1).
- At Step b/, for each $i \in I$ we remove the old event associated to i and insert the one computed at Step a/.
- At Step c/, we retrieve the minimum of all possible times to find out which subprocess i is actually generating a new point.

- At Step d/, we look for the children of i and update I.

- Step e/ depends hugely on the type of process at hand. The apparition of a new point has clear impact on the intensity but this depends on the formula. For instance if the intensity is given by Equation (3.1), we need to shift intensities and update sums (see Section 5 for more details).

More details about the algorithm steps are provided in Section 3.6, which presents in full details the application of the algorithm to the Hawkes case.

Algorithm 14 Local graph algorithm for the simulation of point processes: Application of the simulation activity tracking algorithm [23].

$k \leftarrow 0, t_k \leftarrow 0$
$I \leftarrow \{1, ..., M\}$
while $t_k < T$ **do**
Step a/ **Compute the next possible points** t^i_{next} for each $i \in I$ based on intensity λ_i on $(t_k, +\infty)$
Step b/ **Update** Q with each next possible point t^i_{next} for each $i \in I$
Step c/ **Get next selected point** $t_{k+1} \leftarrow min\{t^i_{next}\}$ and i the **associated node**, updating $Q \leftarrow Q^*$
Step d/ **Find the children of** i **and update** $I \leftarrow ch(i) \cup \{i\}$
Step e/ **Update intensities** $\lambda_j(t)$ for each node $j \in I$ on $(t_{k+1}, +\infty)$
$k \leftarrow k + 1$
end while
return $(t_1, ..., t_{k-1})$ points and associated nodes $(i_1, ..., i_{k-1})$

3.6 Hawkes evaluation

We want to evaluate the complexity of the previous algorithms, but this of course depends on the computational complexity of the conditional intensities associated to each point process. Previous general algorithms for simulating point processes are applied here to non explosive Hawkes processes with piecewise constant interactions with finite support (see Equation (3.1)). In this situation, note that the λ_i's become piecewise constant, so that the complexity for calculating such intensities or updating them will be linked to the number of breakpoints of the corresponding piecewise constant function. Moreover with piecewise constant intensities, one can apply the transformation method directly, so we do not evaluate the complexity of the thinning /rejection step. The general algorithms are specified at data structure level in order to detail the computational complexity of each algorithmic step.

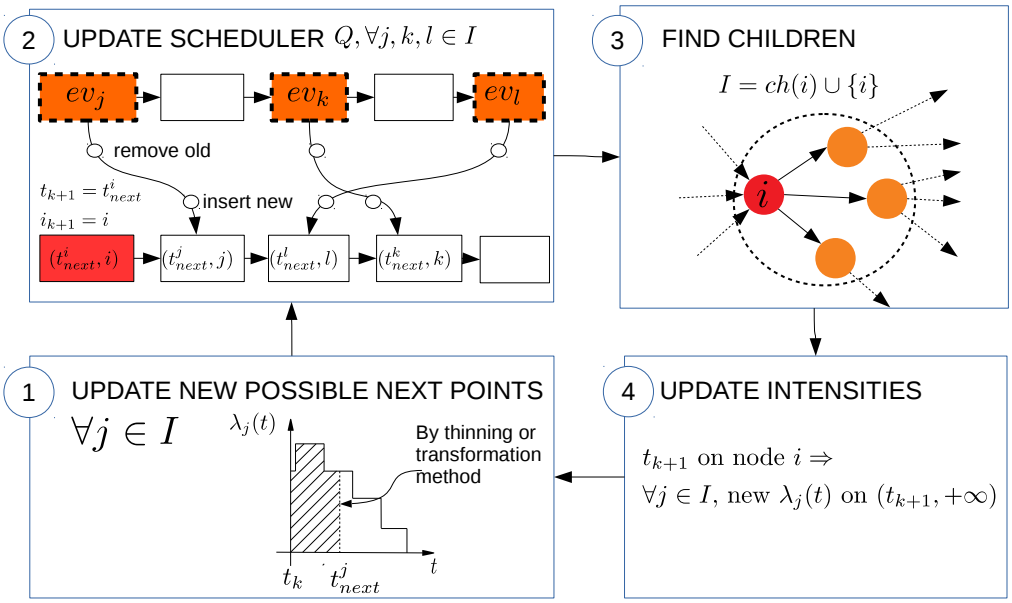


Figure 3.12: Steps of the local graph algorithm for the simulation of point processes. As for figure 3.1, the intensities are piecewise constant.

3.6.1 Notation and data structures

In the following, $\leftrightarrow$ means "corresponds to the mathematical notation". Data structure notation consists of:

- Q: A *scheduler of next point events* $ev_i = (t^i_{next}, i)$, where t^i_{next} is a *possible next point* associated to node i.
- $L[i](\leftrightarrow \lambda_i)$: is the scheduler corresponding to the piecewise constant intensity of node i, The *length of the scheduler* is $L^i_t = length(L[i])$ when $L[i][0].time = t$.
- $h[j][i](\leftrightarrow h_{j\to i})$: is the scheduler corresponding to the piecewise constant interaction $h_{j\to i}$ (with support included in $[0, S]$) from node j to node i. The *maximum number of events in* $h[j][i]$ is noted $A \geq length(h[j][i])$.
- $\overline{L} = L[1] \cup ... \cup L[M]$ $(\leftrightarrow \overline{\lambda}(t) = \sum_{i=1}^{M} \lambda_i(t))$: is a scheduler storing the piecewise sum of all $L[i]$, from node 1 to node M (cf. Figure 3.8).
- $\overline{L}[i] = L[1] \cup ... \cup L[i]$ $(\leftrightarrow \overline{\lambda}_i(t) = \sum_{j=1}^{i} \lambda_j(t))$: is a scheduler storing the partial piecewise sum of the intensities from node 1 to node i. Obviously, $\overline{L}[M] = \overline{L}$.

3.6.2 Algorithm for the transformation method for piecewise constant intensities

Algorithm 15 presents the basic transformation method for piecewise constant conditional intensity functions, here represented by the scheduler Q (cf. Equation 3.2).

Algorithm 15 Function getTnext(Q) with Q a scheduler storing the events corresponding to a piecewise constant intensity trajectory.

1: **function** getTnext(Q)
2: $V \sim \mathcal{U}[0,1]$
3: $integral \leftarrow 0$
4: $t_{next} \leftarrow Q[0].time$
5: $k \leftarrow 0$
6: $val \leftarrow 0$
7: **repeat**
8: $val \leftarrow val + Q[k].value$
9: $integral \leftarrow integral + (Q[k+1].time - Q[k].time) \times val$
10: $k \leftarrow k + 1$
11: **until** $(integral > -log(V)$ or $k = size(Q) - 1)$
12: **if** $integral \leq -log(V)$ **then**
13: $val \leftarrow val + Q[k].value$
14: **end if**
15: **return** $t_{next} \leftarrow Q[k].time - \frac{integral+log(V)}{val}$ // $t_{next} \leftarrow +\infty$ if $val = 0$
16: **end function**

The complexity of the getTnext(Q) operation is $\mathcal{O}(|Q|)$.

3.6.3 Full scan and local graph algorithms

Algorithm 16 is the application of Algorithm 13 to Hawkes processes. The mention to a, b c, d refers to the steps in Algorithm 13. We split step d to lower the complexity. The value T is an arbitrary stopping time, the condition $t_k < T$ ending the main loop in both Algorithm 16 and Algorithm 17 can be changed by whatever condition is deemed more appropriate by the modeller.

Algorithm 16 Full scan algorithm for Hawkes processes

$t_0 \leftarrow 0$
$k \leftarrow 0$
while $t_k < T$ **do**
  Step a/ $\overline{L}[1] \leftarrow L[1]$
  Step a/ **for all** $j \in \{2, ..., M\}$ **do**
  Step a/ Prune intensities $\overline{L}[j] \leftarrow \overline{L}[j-1] \cup L[j]$
  Step b/ **end for**
  Step d1/ $t_{k+1} \leftarrow$ GETTNEXT($\overline{L}[M]$)
  Step d1/ **for all** $j \in \{1, ..., M\}$ **do**
  Step c/ $L[j] \leftarrow L[j]^{t_{k+1}}_{pcw}$
  Step c/ **end for**
  Step c/ $\ell[0] \leftarrow 0$
  Step c/ **for all** $j \in \{1, ..., M\}$ **do**
  Step d2/ compute $\ell[j] = \overline{\lambda}_j(t_{k+1})$ **by** $\ell[j] \leftarrow \ell[j-1] + L[j][0].value$
  Step d2/ **end for**
  Select the associated node i_{k+1} as the only j such that $\frac{\ell[j-1]}{\ell[M]} < V \leq \frac{\ell[j]}{\ell[M]}$ for $V \sim \mathcal{U}[0,1]$
  for all $j \in \{1, ..., M\}$ **do**
  Update intensities $L[j] \leftarrow L[j] \cup h[i_{k+1}][j]_{\rightarrow_{t_{k+1}}}$
  end for
end while
$k \leftarrow k+1$
return points $(t_1, ..., t_{k-1})$ and associated nodes $(i_1, ..., i_{k-1})$

Algorithm 17 is the application of Algorithm 14 to Hawkes processes.

Algorithm 17 Local graph algorithm for Hawkes processes

$I \leftarrow \{1, ..., M\}, k \leftarrow 0$
while $t_k < T$ **do**
  Step a/ **Compute the next point** $t^i_{next} \leftarrow$ GETTNEXT$(L[i])$ of each $i \in I$
  Step b/ $Q \leftarrow Q^* \oplus (t^i_{next}, i)$ of each $i \in I$
  Step c/ $t_{k+1} = Q[0].time$
  Step c/ $i_{k+1} = Q[0].value$
  Step d/ **Find children of** i **and update** $I \leftarrow ch(i_{k+1}) \cup \{i_{k+1}\}$
  Step e/ **for all** $i \in ch(i_{k+1})$ **do**
    Step e/ $L[i] \leftarrow L[i]^{t_{k+1}}_{pcw} \cup h[i_{k+1}][i]_{\rightarrow_{t_{k+1}}}$
  Step e/ **end for**
  Step e/ **if** $i \notin ch(i_{k+1})$ **then**
    Step e/ $L[i] \leftarrow L[i]^{t_{k+1}}_{pcw} \cup h[i_{k+1}][i]_{\rightarrow_{t_{k+1}}}$
  Step e/ **end if**
  $k \leftarrow k + 1$
end while
return $(t_1, ..., t_{k-1})$ points and associated nodes $(i_1, ..., i_{k-1})$

3.6.4 Complexities of both algorithms

If A (the number of breakpoints to describe the interaction functions $h_{j\rightarrow i}$) and S (the support of the $h_{j\rightarrow i}$'s) are true constants, assumed to be of order 1 in the sequel, the size of the different schedulers that are used in the previous algorithms are most of the time random and changing step after step. They depend in particular on the number of points of N_j appearing in the interaction range that is $N_j([t - S, t))$. To evaluate further the order of such a random quantity, we know that a stationary Hawkes process has a mean intensity vector $m = (m_1, ..., m_M)^T$ (see [11]):

$$m = (I_M - H)^{-1}\nu, \tag{3.3}$$

with $\nu = (\nu_1, ..., \nu_M)^T$, I_M the identity matrix of size M and $H = (\int_0^{+\infty} h_{j\rightarrow i}(x)\ \mathrm{d}x)_{i,j=1,...,M}$. Note that the linear Hawkes process is explosive when the spectral radius of H is strictly larger than 1 (we refer the reader to [5] and [21] for demonstrations even in the non-linear cases). When the spectral radius is strictly less than 1, the non explosive Hawkes process, with no points before time 0, has always less points than the stationary version. Therefore,

$$\mathbb{E}(N_j([t - S, t))) \leq m_j S \tag{3.4}$$

with m_j given by Equation (3.3).

Moreover, the local independence graph for Hawkes process is completely equivalent to the graph with edge $j \to i$ if and only if $h_{j\to i}$ is non zero. The corresponding adjacency matrix is denoted $R = (\mathbf{1}_{\int h_{j\to i} \neq 0})$.

At time t, the scheduler $L[i]$ describes the piecewise constant conditional intensity $\lambda_i(.)$ on $[t, +\infty)$ in absence of new points after t. The number of breakpoints of $L[i]$ is denoted L_t^i. But (3.1) can be rewritten as

$$\lambda_i(t) = \nu_i + \sum_{j \in pa(i)} \sum_{T \in N_j, T \in [t-S,t)} h_{j\to i}(t - T)$$

So we can first note that L_t^i and therefore its expectation $\mathscr{L}_i = \mathbb{E}(L_t^i)$ are always larger than 1 because the scheduler $L[i]$ is at least of size 1. Moreover this piecewise constant function has potential breakpoints at all $T + a$, for $T \in N_j, T \in [t - S, t)$, and a breakpoints of $h_{j\to i}$.

Therefore we can compute the order of magnitude of L_t^i, which is the length of the scheduler associated to $\lambda_i(t)$, by

$$L_t^i = \mathcal{O}\left(1 + A \sum_{j \in pa(i)} N_j([t - S, t)\right)$$

where $\mathcal{O}$ means that there exists an absolute positive constant C such that

$$L_t^i \leq C\left(1 + A \sum_{j \in pa(i)} N_j([t - S, t)\right).$$

In expectation, this gives, thanks to (3.4) and since $AS = \mathcal{O}(1)$,

$$\mathscr{L} = \mathcal{O}(1 + Rm) = \mathcal{O}\big(1 + R(I_M - H)^{-1}\nu\big) \tag{3.5}$$

with $\mathscr{L} = (\mathscr{L}_i)_{i=1,\dots,M}$, the notation $\mathcal{O}$ being understood coordinate by coordinate.

Now we can evaluate the (mean) complexity of both algorithms, replacing L_t^i by $\mathscr{L}_i$ thanks to the respective complexities of each operation on the schedulers, see Section 3.4.

Full-Scan Algorithm Step a/ has a complexity of

$$\mathcal{O}\left(\sum_{j=1}^{M} \mathscr{L}_i \log(\sum_{i=1}^{j} \mathscr{L}_i)\right) = \mathcal{O}\big(|\mathscr{L}|_1 \log |\mathscr{L}|_1\big)$$

with $|\mathscr{L}|_1 = \mathscr{L}_1 + ... + \mathscr{L}_M \geq M$ Step b/ has complexity $\mathcal{O}\big(|\mathscr{L}|_1\big)$, as well as Step d1/. Step c/ has complexity $\mathcal{O}(M) \leq \mathcal{O}\big(|\mathscr{L}|_1\big)$. Step d2/ has complexity

$$\mathcal{O}\left(\sum_{j=1}^{M}(A\log(\mathscr{L}_i) + A + \mathscr{L}_i)\right) = \mathcal{O}\big(|\mathscr{L}|_1 \log|\mathscr{L}|_1\big)$$

So globally one iteration of the full scan algorithm has a complexity of the order

$$\mathcal{O}\big(|\mathscr{L}|_1 \log|\mathscr{L}|_1\big) = \mathcal{O}\big((M + |Rm|_1)\log(M + |Rm|_1)\big).$$

Therefore since the mean total number of iterations of this algorithm is also the mean total number of points produced on $[0, T]$, that is $T|m|_1$, the full-scan algorithm should have the following mean complexity

$$\mathcal{O}\big(T|m|_1(M + |Rm|_1)\log(M + |Rm|_1)\big). \tag{3.6}$$

As expected, the complexity is linear with the duration T of the simulation. Moreover this complexity heavily depends on the whole set of parameters (type of graph, strength of the interaction functions etc), because in particular these parameters affect the number of points that have to be produced. So for very unbalanced networks where $|m|_1 = \mathcal{O}(1)$ (if for instance only one node in the whole network is clearly active and the others almost silent), the complexity seems to be of order $\mathcal{O}(TM\log(M))$. But these very unbalanced networks are not the most usual. Let us look now at more balanced networks. Let us assume that all the m_j's are roughly the same and are of order 1 (no really small m_j) and that the number of parents of a given node is bounded by d, this give us a complexity of

$$\mathcal{O}\big(TM^2 d\log(dM)\big).$$

So up to the log factor, if the network is sparse but balanced, the complexity is quadratic in the number of nodes of the network. If the network is a full complete graph, the complexity is cubic in M.

Local graph algorithm As before we need to evaluate first the complexity of one iteration of the algorithm. But because I is chosen at step d/ and the size of I impacts the complexity of steps a/b/ and e/, we choose to evaluate the complexity of an iteration which starts with e/ and then does a/b/ c/ and d/, so that that until d/ the set I is the same.
If the node $i_{k+1} = j$, then the complexity of step e/ is

$$\mathcal{O}\left(\mathscr{L}_j + \sum_{i\in ch(j)}(\mathscr{L}_i + A + A\log(\mathscr{L}_i))\right) = \mathcal{O}\left(\mathscr{L}_j + \sum_{i\in ch(j)}(\mathscr{L}_i + \log(\mathscr{L}_i))\right).$$

The complexity of step a/ is

$$\mathcal{O}\left(\mathscr{L}_j + \sum_{i \in ch(j)} \mathscr{L}_i\right).$$

The complexity of step b/ is

$$\mathcal{O}\left(\log(M) + \sum_{i \in ch(j)} \log(M)\right).$$

Steps c/ and d/ have complexity $\mathcal{O}(1)$.

So for one iteration "e/a/b/c/d/" after a point on node j, the complexity is

$$\mathcal{O}\big(\mathscr{L}_j + \log(M) + [R'(\mathscr{L} + \log(M)\,\mathbf{1})]_j\big),$$

with R' the transpose of R and $\mathbf{1}$ the vector of size M full of ones.

The main point is that a point in N_j is appearing in average at most only Tm_j times during the simulation since m_j is the mean intensity of N_j, which leads us to a global complexity of

$$\mathcal{O}\big(Tm'\mathscr{L} + T\log(M)|m|_1 + Tm'R'[\mathscr{L} + \log(M)\mathbf{1}]\big) = T\mathcal{O}\big(m'Rm + \log(M)|m|_1 + m'R'Rm + \log(M)|Rm|_1\big). \tag{3.7}$$

As before this is linear in the duration of the simulation T and depends heavily on the parameters. But the complexity is much lower. Indeed, for very unbalanced networks where only one node is really active, the complexity logarithmic in M. For balanced networks where the m_j's are roughly the same and if the number of children of a given node, as well as the number of parents is bounded by d, then we get a complexity of

$$\mathcal{O}\big(TMd[d + \log(M)]\big),$$

For sparse balanced graphs, we therefore get a complexity which is linear in M up to logarithmic factors. The gain is clear with respect to the full scan algorithm. For complete graphs, we also get a cubic complexity in terms of M, as the full scan algorithm but without logarithmic factors.

So at least theoretically speaking, it seems that the local graph algorithm is always a better choice than the full-scan algorithm, with a clear decrease of complexity from quadratic to linear in the number of nodes for balanced sparse graphs.

3.7 Numerical experiments

This section is devoted to two main problems: statistically proving that both algorithm (full scan and local graph) indeed simulate a Hawkes process and asserting that the local graph algorithm clearly outperforms the full scan algorithm.

3.7.1 Hardware and software specifications

The main simulations have been performed on 5 nodes of a Symmetric MultiProcessing (SMP), i.e., shared memory, computer1. Each of this computational nodes has up to 20 physical cores (210), 25 MB of cache memory and 62.5 GB of RAM. The processors are Intel(R) Xeon(R)CPU E5-2670 (v0 and v2) at 2.60 GHz. The statistical analysis required more RAM, so we used another type of node, which has 770GB of RAM, 25MB of cache memory, 20 physical cores (210), each processor being an Intel(R) Xeon(R)CPU E5-2687W v3 at 3.10GB. The algorithms were implemented in C++ programming language (2011 version). No other external libraries were used for the simulator, which is compiled using gcc 4.7. The plots and statistical analyses were obtained using using R software (v3.6), part of it using the UnitEvent package (v0.0.5).

3.7.2 Statistical analysis

We generated an Erdös-Rényi network of 100 nodes with connection probability $p = 1/100$, that is fixed for the rest of the statistical analysis. When an edge $j \to i$ is in the graph, we associate it to an interaction function $t \mapsto h_{j\to i}(t) = 5 \cdot \mathbb{1}_{t\in[0,0.02]}$. The spontaneous parameters ν_i are all fixed to 10. Out of this multivariate Hawkes process, we focus on two nodes a and b. The node a is fully disconnected, meaning the corresponding process should be an homogeneous Poisson process of rate 10. The node b is the one with the largest number of parents (4 parents).

Time transformation In [**Ogata_test**], Ogata derives methodological benchmarks to assess if the data are obeying a point process with a given intensity, and in particular Hawkes processes. This is based on the time-rescaling theorem (see for instance [7]), which says that if λ_s is the conditional intensity of the point process N and if $\Lambda(t) = \int_0^t \lambda_s ds$, then the points $\tilde{N} = \{\Lambda(T), T \in N\}$ form an homogeneous Poisson process of rate 1. Ogata proposed the following method to test that a given point process has intensity given by λ_s

- Apply the time-rescaling transformation. This leads to a point process $\tilde{N}$.
- Test that the consecutive delays between points of $\tilde{N}$ obeys an exponential distribution of rate 1, for instance by Kolmogorov-Smirnov test (**Test 1**)

- Test that the points of $\tilde{N}$ themselves are uniformly distributed, for instance by Kolmogorov-Smirnov test (**Test 2**).

- Test that the delays between points of $\tilde{N}$ are independent, for instance by checking that the autocorrelation between delays with a certain lag are null (**Tests 3**). We performed them up to lag 9.

We simulated the multivariate Hawkes process on $[0, T]$ with $T = 150$ and we applied the previous tests to node a and node b.

Table 3.1: Table of the p-values of a Kolmogorov-Smirnov test (for uniformity) applied to the p-values obtained with tests 1, 2 and 3 for 1000 independent simulations of the same Hawkes point processes (with the same underlying graph).

	FULL-SCAN		LOCAL-GRAPH	
	Node a	Node b	Node a	Node b
Test 1	0.5384525	0.1491268	0.0594925	0.86789804
Test 2	0.6008973	0.2462138	0.1819709	0.99025263
Test 3 with lag 1	0.1602718	0.1781804	0.4385096	0.92162419
Test 3 with lag 2	0.7498109	0.9038829	0.6954876	0.90558993
Test 3 with lag 3	0.5604420	0.7220130	0.4144515	0.77140051
Test 3 with lag 4	0.7003987	0.1838913	0.4367523	0.83833821
Test 3 with lag 5	0.9960351	0.4009543	0.3740874	0.14749913
Test 3 with lag 6	0.1883506	0.1246654	0.4387684	0.12202262
Test 3 with lag 7	0.1259022	0.8588754	0.9114556	0.47030751
Test 3 with lag 8	0.8848928	0.9720601	0.5200698	0.03765871
Test 3 with lag 9	0.2278844	0.3880436	0.5042846	0.92768290

If we have simulated indeed the correct Hawkes processes for the processes associated to node a and b, the p-values should be uniform. So we performed 1000 simulations of the same Hawkes process (with the same underlying graph) but with different pseudorandom generator seeds for the simulation of the points themselves. We can visually check that they are indeed uniform by seeing diagonals for their cumulative distribution functions (see Figures 3.13 and 3.14). In order to confirm this qualitative result with a more quantitative one, the p-values for the three tests 1, 2 and 3 are independently tested for uniformity with another Kolmogorov-Smirnov test. The resulting p-values are displayed in Table 3.1.

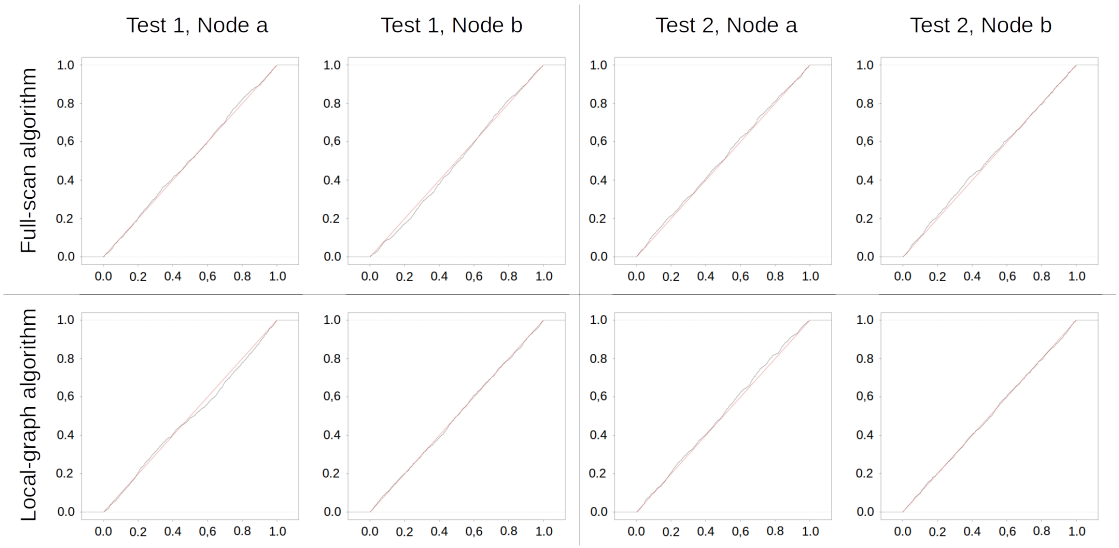


Figure 3.13: Cumulative distribution functions of the p-values of Test 1 and 2. In columns the test and node, in rows the algorithms (full scan then local graph)

Martingale properties Another very important property of the Hawkes process is that $t \mapsto N_t - \Lambda_t$ is a martingale and this property remains true if we integrate with respect to a predictable process. So for each node a or b, we can compute

$$X^k = \int_0^T \psi_t^k (dN_t - d\Lambda_t),$$

for $\psi_t^1 = 1$ or $\psi_t^{2j} = N_j([t-0.02, t)]$ or $\psi_t^{2j+1} = N_j([t-0.04, t-0.02)]$. If the martingales properties are true, then the variable X^k for each k should be centered around 0. We also expect eventually different behaviors, when $k = 1$, which corresponds to the spontaneous part or when $k = 2j$ or $2j+1$ for a node j which is connected to the node of interest or disconnected from the node of interest. We simulated the network 40 times on $[0, T]$ with $T = 20$ and reported the X^k. We see on Figure 3.15 and 3.16 that the variables X^k are indeed centered in both cases as expected. So we can conclude that both algorithms indeed simulate the given Hawkes process.

3.7.3 Performance

We want to assess the performances of both algorithm in the main interesting case: sparse balanced networks. To do so, we took three different topologies of graphs:

- Erdös-Rényi: a topology model where each edge has a probability p of being present or absent, independently of the other edges. We took p in $\{0, \frac{1}{M}, \frac{2}{M}, \dots, \frac{(ln(M)-1)}{M}\}$ for M the

number of nodes. These choices for p ensure a sparse graph with roughly speaking $d = pM$ parents and children for each node.

- Cascade: a classical topology model where each node has exactly one parent and one child (except for two nodes, start and end, that have respectively no parent and one child, and one parent and zero child), and there are no cycles in the network. There is only one graph per number M.

- Stochastic-Block: it is an Erdös-Rényi by block. In our setting the nodes are partitioned in two blocks and the matrix gives the probability of inter-block connection of intra-block connection (see Table 3.2).

Table 3.2: The three bloc sizes vectors (first line) and the three probability matrices (second line) used for the simulations.

Block sizes	$\begin{pmatrix} \frac{M}{2} & \frac{M}{2} \end{pmatrix}$	$\begin{pmatrix} \frac{M}{2} & \frac{M}{2} \end{pmatrix}$	$\begin{pmatrix} \ln M & M - \ln M \end{pmatrix}$
Probability matrices	$\begin{pmatrix} \frac{2}{M}\ln\frac{M}{2} & 0 \\ 0 & \frac{2}{M}\ln\frac{M}{2} \end{pmatrix}$	$\begin{pmatrix} 0 & \frac{2}{M}\ln\frac{M}{2} \\ \frac{2}{M}\ln\frac{M}{2} & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & \frac{\ln(\lceil \ln M \rceil)}{\lceil \ln M \rceil} \\ \frac{\ln(M-\lceil \ln M \rceil)}{(M-\lceil \ln M \rceil)} & 0 \end{pmatrix}$

Each graph was generated using a different pseudorandom generator seed. Each existing edge $j \to i$ is associated with an interaction function $t \mapsto h_{j\to i}(t) = 5 \cdot \mathbb{1}_{t\in[0,0.02]}$. We computed the largest eigen-value of the corresponding matrix H. If it is larger than 1, this graph should be discarded. To force the balance of the network, we decided to take $m = (10, ..., 10)$ and compute the ν_i's by $\nu = (I_M - H)m$. It may happen that some of the ν_i's become negative. These graphs should be also discarded, as by construction the conditional intensity $\lambda(t)$ of a Hawkes point process must remain positive at all time, and so it is the case for the background intensity ν_i (see [6] for reference). Because of the parameter values, especially the interaction functions, no graph was discarded here. A total of $2890 = 1770 + 280 + 840$ (Erdös-Rényi + Cascade + Stochastic-Block) graphs was obtained with $M = \{10, 20, \dots, 100\} \cup \{150, 200, \dots, 500\} \cup \{600, 1100, \dots, 5100\}$ for the local-graph algorithm, and $M = \{10, 20, \dots, 100\} \cup \{150, 200, \dots, 500\}$ for the full-scan algorithm. Once the parameters of the Hawkes process are fixed, we simulated 10 times each process on $[0, T]$ with $T = 10$, each simulation with a different generator seed.

Figure3.17 shows that the theoretical complexities of both full scan and local graph algorithms are equivalent to their actual execution times.

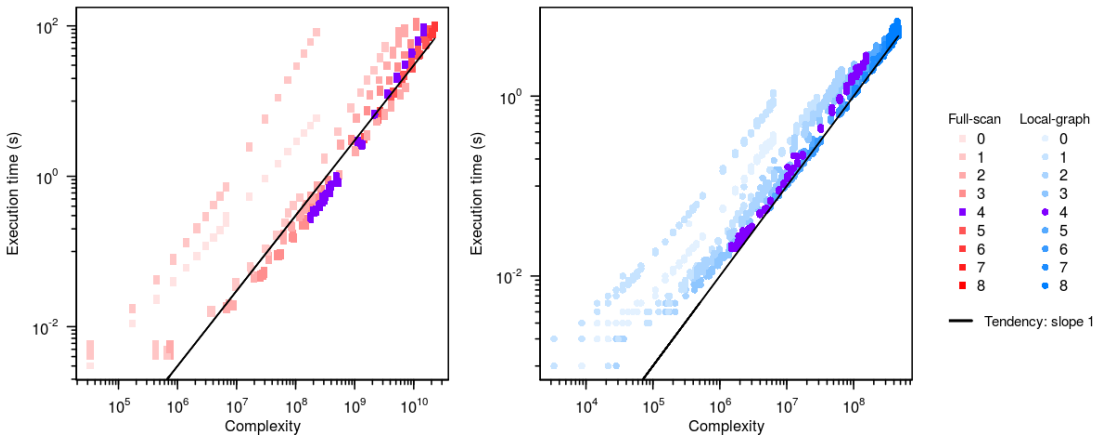


Figure 3.17: Three topologies together: the execution time (vertical axis, log-scaled) and the theoretical complexity (horizontal axis, log-scaled), for the full scan algorithm (red squares, left part) and the local-graph algorithm (blue circles, right part). A line of slope 1 is displayed in black, on both scatter plots, showing the equivalence. The colour gradient represents similar values of the mean number of connections per process. A particular value (mean of 4 children per process) is emphasised with a violet tone. The number of nodes is $M = \{10, 20, \dots, 100\} \cup \{150, 200, \dots, 500\}$ for the full-scan algorithm and $M = \{10, 20, \dots, 100\} \cup \{150, 200, \dots, 500\} \cup \{600, 1100, \dots, 5100\}$ for the local graph algorithm.

Figure 3.18 shows that the execution time is quadratic for the full scan algorithm and linear behaviour for the local graph algorithm. For example, when the local graph algorithm is executed in less than 10s for more than 5000 nodes, the execution of the full scan algorithm takes about 100s for 500 nodes. The local graph algorithm clearly outperforms the full scan algorithm.

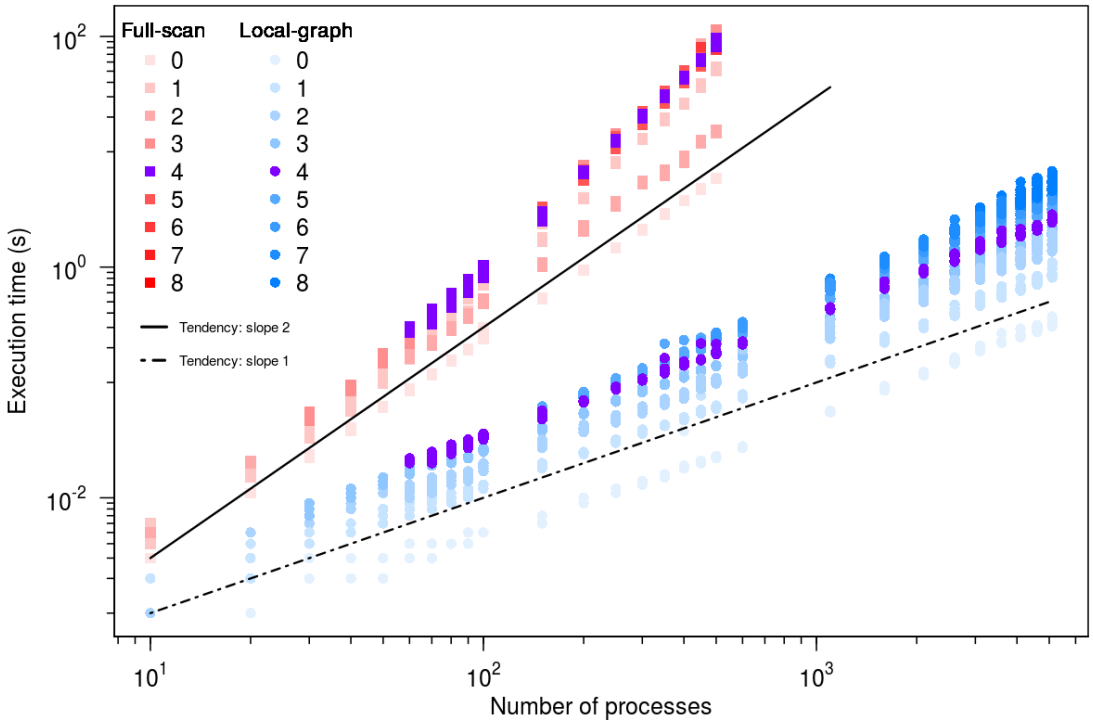


Figure 3.18: Three topologies together: the execution time (vertical axis, log-scaled) and the theoretical complexity (horizontal axis, log-scaled), for the full scan algorithm (red squares, left part) and the local-graph algorithm (blue circles, right part). A line of slope 2 is displayed in black, on the scatter plot for the full scan algorithm, and a line of slope 1 for the local graph algorithm. The colour gradient represents similar values of mean numbers of connections per process. A particular value (mean of 4 children per process) is emphasised with a violet tone. The number of nodes is $M = \{10, 20, \dots, 100\} \cup \{150, 200, \dots, 500\}$ for the full-scan algorithm and $M = \{10, 20, \dots, 100\} \cup \{150, 200, \dots, 500\} \cup \{600, 1100, \dots, 5100\}$ for the local graph algorithm.

3.8 Conclusion

We presented a new discrete event simulation for point processes: the local graph algorithm, aiming at tracking only the nodes changing state in the network, only updating their children (based on the local independence graph hypothesis [13]). The computational complexity reduction of the local graph algorithm with respect to the full scan algorithm (an adaptation of Ogata's algorithm [26]) is from M^2 to M. Although there was no simulation algorithm able to simulate large point process networks, the local graph algorithm now opens new perspectives for simulating such networks. Especially, based on the local graph generation, an interesting perspective concerns the memory reduction. Instead of statically storing the whole network topology in memory at the

beginning of the simulation, only the local graphs corresponding to the children of changing state nodes could be dynamically generated during the simulations [17]. Based on time asynchrony, this corresponds to asynchronous dynamic structure changes in discrete event systems [24]. Generating only local graphs with respect to the whole network graph should allow simulating very large networks. The same complexity reduction order is expected. However, at execution time level, the cost of re-generating the local graphs will have to be taken into account.

The network structure plays a central role in the arguments. While we assume that all processes in the population are of the same type, the connectivity between the processes in the population is not homogeneous. Each process in the population of N nodes receives input from C randomly selected processes in the population. Sparse connectivity means that the ratio $\delta = \frac{C}{N} \ll 1$ is a small number. One can ask if it is this realistic. In the context of the human brain, a typical pyramidal neuron in the cortex receives several thousand synapses from presynaptic neurons while the total number of neurons in the cortex is much higher [27]. Thus globally the cortical connectivity $\frac{C}{N}$ is low. On the other hand, we may concentrate on a single column in visual cortex and define, e.g., all excitatory neurons in that column as one population. We estimate that the number N of neurons in one column is below ten thousand. Each neuron receives a large number of synapses from neurons within the same column. In order to have a connectivity ratio of 0.1, each neuron should have connections to about a thousand other neurons in the same column. [15]. In the brain, last estimations consist of 86 billions of neurons [3], each neuron having around $7'000$ connections. *Either for the overall brain or for a single column of the visual cortex, the hypothesis of sparse connectivity of the network remains valid.* This work thus allows achieving grounded stochastic simulations of the neuronal functional interactions in parts of the human brain.

Acknowledgement

For the SMP simulations we would like to deeply thank the LIMOS CNRS laboratory, from the University of Clermont-Auvergne, which graciously provided access. In particular we would like to thank their current administrators, Hélène Toussaint, William Guyot-Lénat and Boris Lonjon, for their valuable help.

This work was supported by the French government, through the UCA^{Jedi} and 3IA Côte d'Azur Investissements d'Avenir managed by the National Research Agency (ANR-15- IDEX-01 and ANR-19-P3IA-0002) and by the interdisciplinary Institute for Modeling in Neuroscience and Cognition (NeuroMod) of the Université Côte d'Azur.

References for Chapter 3

1. P. Andersen, O. Borgan, R. Gill, and N. Keiding. *Statistical Models Based on Counting Processes*. Springer, 1996.
2. M. Barrio, K. Burrage, A. Leier, and T. Tian. "Oscillatory Regulation of hes1: Discrete Stochastic Delay Modelling and Simulation". *PLoS Computational Biology* 2:9, 2006, p. 1017.
3. C. von Bartheld, J. Bahney, and S. Herculano-Houzel. "The search for true numbers of neurons and glial cells in the human brain: A review of 150 years of cell counting". *Journal of Comparative Neurology* 524:18, 2016, pp. 3865–3895.
4. A. Bouchard-Côté, S. J. Vollmer, and A. Doucet. "The Bouncy Particle Sampler: A Non-Reversible Rejection Free Markov chain Monte Carlo Method". *Journal of the American Statistical Association* 113, 2018, pp. 855–867.
5. P. Bremaud and L. Massoulie. "Stability of Nonlinear Hawkes Processes". *The Annals of Probability* 24:3, 1996, pp. 1563–1588. ISSN: 00911798. URL: http://www.jstor.org/stable/2244985.
6. P. Brémaud. *Point Processes and Queues*. Springer series in statistics. Springer New York, 1981. ISBN: 9781468494778. URL: https://books.google.fr/books?id=lo-YZwEACAAJ.
7. E. N. Brown, R. Barbieri, V. Ventura, R. E. Kass, and L. Frank. "The Time-Rescaling Theorem and Its Application to Neural Spike Train Data Analysis". *Neural Computation* 14:2, 2002, pp. 325–346.
8. E. Brown, R. Barbieri, V. Ventura, R. Kass, and L. Frank. "The Time-Rescaling Theorem and Its Application to Neural Spike Train Data Analysis". *Neural Computation* 4:2, 2006, pp. 325–346.
9. J. Cha and M. Finkelstein. *Point Processes for Reliability Analysis*. Springer, 2018.
10. J. Chevallier, M. Cáceres, M. Doumic, and P. Reynaud-Bouret. "Microscopic approach of a time elapsed neural model". *Mathematical Models and Methods in Applied Sciences* 25:14, 2015, pp. 2669–2719.
11. D. J. Daley and D. Vere-Jones. *An introduction to the theory of point processes. Vol. I. Probability and its Applications*. 2003.
12. A. Dassios and H. Zhao. "Exact simulation of Hawkes process with exponentially decaying intensity". *Electronic Communications in Probability* 18:62, 2013.
13. V. Didelez. "Graphical models of markes point processes based on local independence". *J.R. Statist. Soc. B* 70:1, 2008, pp. 245–264.

14. J. Doob. "Markoff chains – Denumerable case". *Transactions of the American Mathematical Society* 58:3, 1945, pp. 455–473.

15. W. Gerstner and W. Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

16. D. Gillespie. "Exact Stochastic Simulation of Coupled Chemical Reactions". *he Journal of Physical Chemistry* 81:25, 1977, pp. 2340–2361.

17. P. Grazieschi, M. Leocata, C. Mascart, J. Chevallier, F. Delarue, and E. Tanré. "Network of interacting neurons with random synaptic weights". *ESAIM: ProcS* 65, 2019, pp. 445–475. DOI: 10.1051/proc/201965445. URL: https://doi.org/10.1051/proc/201965445.

18. D. Heger. "A Disquisition on the Performance Behavior of Binary Search Tree Data Structures". 5:5, 2004, pp. 67–75. URL: https://web.archive.org/web/20140327140251/http://www.cepis.org/upgrade/files/full-2004-V.pdf.

19. S. Herculano-Houzel and R. Lent. "Isotropic Fractionator: A Simple, Rapid Method for the Quantification of Total Cell and Neuron Numbers in the Brain". *Journal of Neuroscience* 25:10, 2005, pp. 2518–2521. ISSN: 0270-6474. DOI: 10.1523/JNEUROSCI.4526-04.2005. eprint: https://www.jneurosci.org/content/25/10/2518.full.pdf. URL: https://www.jneurosci.org/content/25/10/2518.

20. P. Lewis and G. Shedler. *Simulation of nonhomogeneous Poisson processes*. Technical report. Naval Postgraduate School, Monterey, California, 1978.

21. I. Mastromatteo, E. Bacry, and J.-F. ç. Muzy. "Linear processes in high dimensions: Phase space and critical properties". *Phys. Rev. E* 91, 4 2015, p. 042142. DOI: 10.1103/PhysRevE.91.042142. URL: https://link.aps.org/doi/10.1103/PhysRevE.91.042142.

22. M. Mesarovic and Y. Takahara. *General systems theory: mathematical foundations*. Vol. 113. Academic press, 1975.

23. A. Muzy. "Exploiting Activity for the Modeling and Simulation of Dynamics and Learning Processes in Hierarchical (Neurocognitive) Systems". *Computing in Science Engineering* 21:1, 2019, pp. 84–93. ISSN: 1558-366X. DOI: 10.1109/MCSE.2018.2889235.

24. A. Muzy and B. P. Zeigler. "Specification of dynamic structure discrete event systems using single point encapsulated control functions". *International Journal of Modeling, Simulation, and Scientific Computing* 5:03, 2014, p. 1450012.

25. J.-F. Muzy, E. Bacry, S. Delattre, and H. M. "Modelling microstructure noise with mutually exciting point processes". *Quantitative Finance* 13:1, 2013, pp. 65–77.

26. Y. Ogata. "On Lewis' simulation method for point processes". *IEEE Transaction on Information Theory* 27:1, 1981, pp. 23–31.

27. B. Pakkenberg, D. Pelvig, L. Marner, M. Bundgaard, H. Gundersen, J. Nyengaard, and L. Regeur. "Aging and the human neocortex". *Experimental Gerontology* 38:1, 2003. Proceedings of the 6th International Symposium on the Neurobiology and Neuroendocrinology of Aging, pp. 95–99. ISSN: 0531-5565. DOI: https://doi.org/10.1016/S0531-5565(02)00151-1. URL: http://www.sciencedirect.com/science/article/pii/S0531556502001511.

28. E. Peters and G. de With. "Rejection-free MonteCarlo sampling for general potentials". *Physical Review E* 85:026703, 2012.

29. P. Reynaud-Bouret, V. Rivoirard, and C. Tuleau-Malot. "Inference of functional connectivity in Neurosciences via Hawkes processes". In: 1st IEEE Global Conference on Signal and Information Processing, Austin, Texas. 2013.

30. P. Reynaud-Bouret and S. Schbath. "Adaptive estimation for Hawkes processes; application to genome analysis". *Annals of Statistics* 38:5, 2010, pp. 2781–2822.

31. K. Tocher. *PLUS/GPS III Specification*. Technical report. Sheffield: United Steel Companies Ltd, Department of Operational Research, 1967.

32. D. Vere-Jones and T. Ozaki. "Some examples of statistical estimation applied to earthquake data." *Ann. Inst. Statist. Math.* 34:B, 1982, pp. 189–207.

33. B. Zeigler. *Theory of Modelling and Simulation*. A Wiley-Interscience Publication. John Wiley, 1976. ISBN: 9780471981527. URL: https://books.google.fr/books?id=MZQAAAAMAAJ.

34. B. Zeigler, A. Muzy, and E. Kofman. *Theory of Modeling and Simulation: Discrete Event & Iterative System Computational Foundations*. Academic Press, 2018.

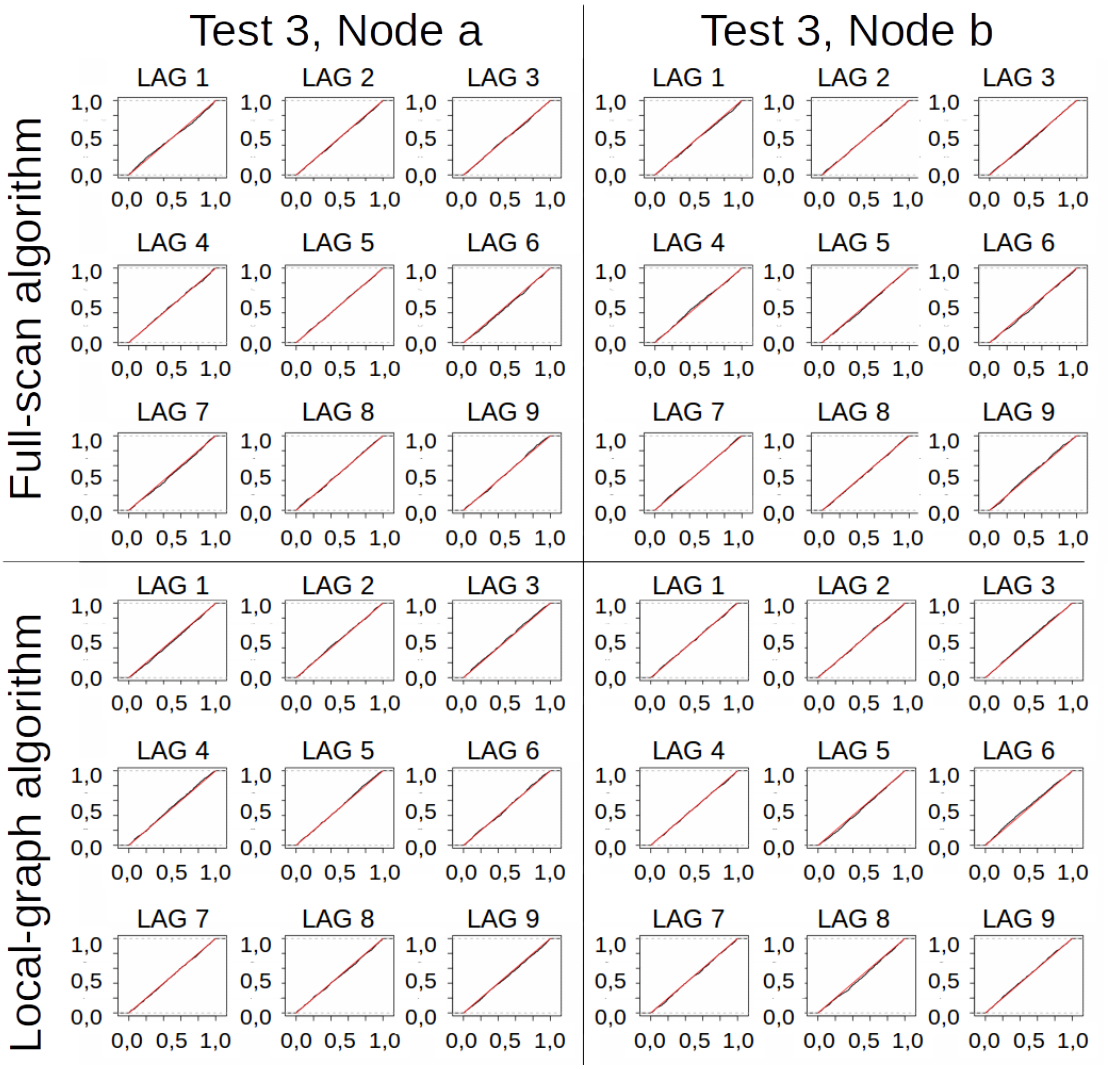


Figure 3.14: Cumulative distribution functions of the p-values of Test 3. In columns the node, in rows the algorithm (full scan then local graph)

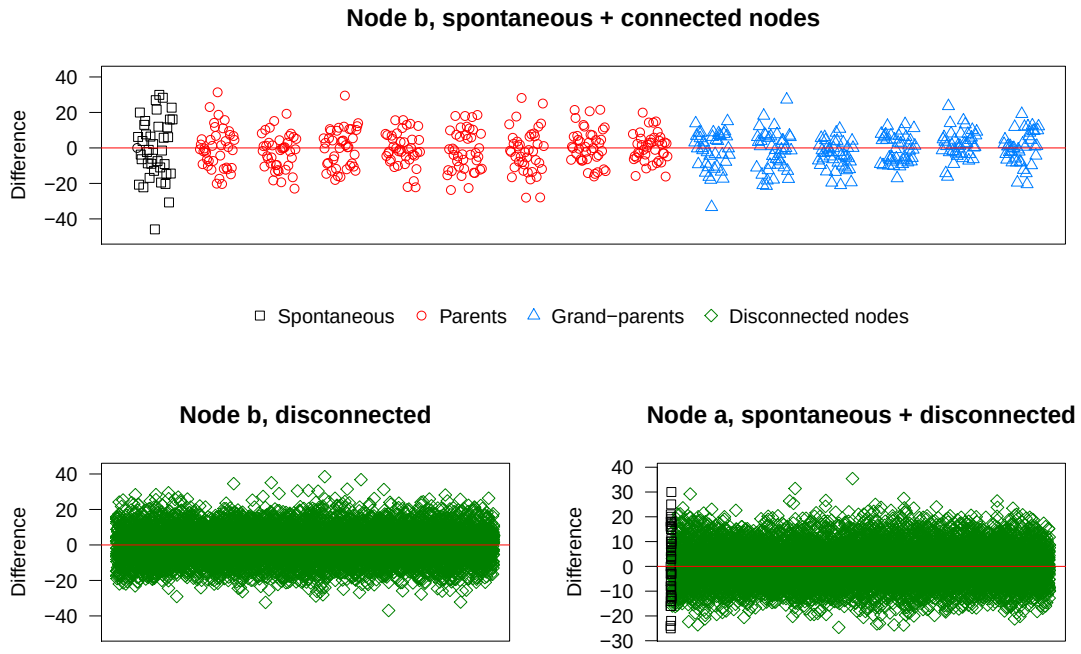


Figure 3.15: Full scan algorithm: verifying the Martingale property for Nodes a and b. The black points represent the X^1 (spontaneous), then for the nodes connected to Node b, X^{2j} and X^{2j+1} are displayed in red. In blue are the X^k and X^{k+1} (still for Node b) from Node b's grand-parents to Node b's parents. Finally the two green scatter plots show the Nodes not disconnected from b and a respectively.

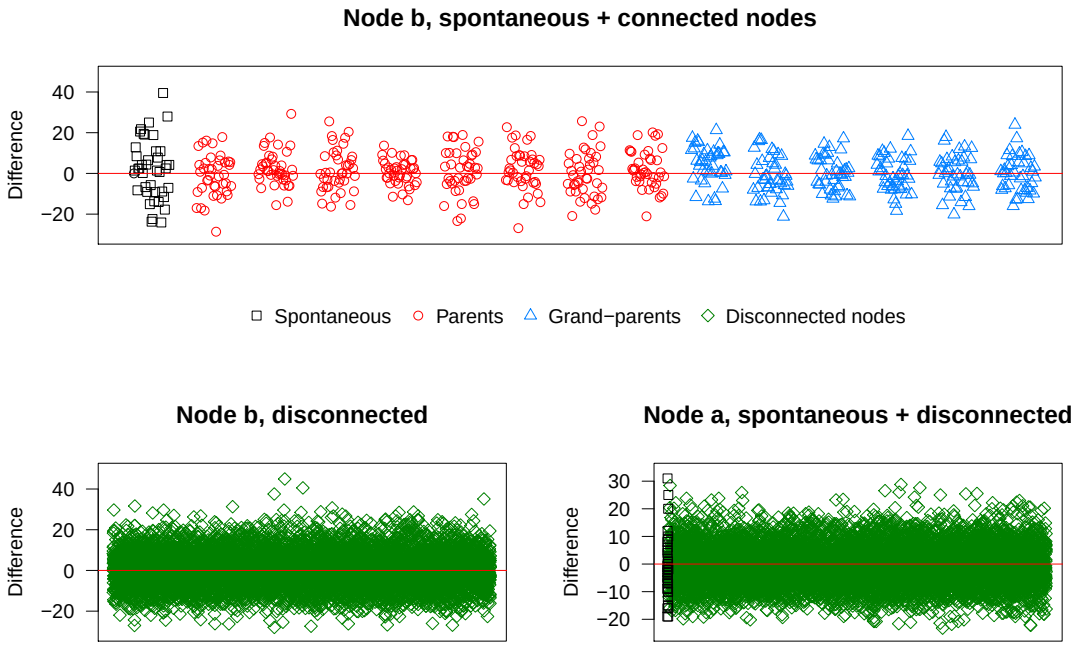


Figure 3.16: Local-graph algorithm: verifying the Martingale property for Nodes a and b. The black points represent the X^1 (spontaneous), then for the nodes connected to Node b, X^{2j} and X^{2j+1} are displayed in red. In blue are the X^k and X^{k+1} (still for Node b) from Node b's grand-parents to Node b's parents. Finally the two green scatter plots show the Nodes not disconnected from b and a respectively.

4 Article 3: Simulation scalability of large brain neuronal networks thanks to time asynchrony

Written by C. Mascart, G. Scarella, A. Muzy, P. Reynaud-Bouret

In this third contribution to the field of computational neuroscience, the algorithm developed in chapter 3 is used for simulating very large network of point processes, with a size of the order of magnitude of the number of neurons in a small mammalian brain. This discrete-event algorithm is advantageous in cases where the mean spiking rate of neurons (or average event rate in the system in general) is small compared to the number of neurons (or components of the system). Indeed, simulation algorithms based on discrete-time management need to use a small time increment to mitigate numerical differentiation errors, which is detrimental to performances. Furthermore, when the spike rate is low, most of the updates of the state of any neurons will leave the state unchanged because of the low event rate.

We show that our algorithm allows the simulation of large-scale neural networks in reasonable time on any modern laptop, without the need for multiprocessor architecture or GPU. We explored the effect of the variation of the connectivity of the neural network on the execution time, and develop how to parametrize the model to guarantee nonexplosivity.

Status: Submitted

Abstract

We present here a new algorithm based on a random model for simulating efficiently large brain neuronal networks. Model parameters (mean firing rate, number of neurons, synaptic connection probability and postsynaptic duration) are easy to calibrate further on real data experiments. Based on time asynchrony assumption, both computational and memory complexities are proved to be theoretically linear with the number of neurons. These results are experimentally validated by sequential simulations of millions of neurons and billions of synapses in few minutes on a single processor desktop computer.

4.1 Introduction

There are more and more vast research projects, whose aim is to simulate brain areas or even complete brains to better understand the way it works. Let us cite for instance: the Human Brain Project [1] in Europe, the Brain Mapping by Integrated Neurotechnologies for Disease Studies (Brain/MINDS) [7] in Japan or the Brain Initiative [25] in the United-States. Several approaches are feasible. There is the biochemical approach [34], which is doomed for systems as complex as the brain. A more biophysical approach has been investigated, see for instance [14], where cortical barrels have been successfully simulated, but are limited to about 10^5 neurons. However, the human brain contains about 10^{11} neurons whereas a small monkey, like marmosets [7], has already 6×10^8 neurons [22] and a bigger monkey, like a macaque, has 6×10^9 neurons [22].

To simulate such huge networks, models reduction have to be made. In particular, a neuron has no more physical shape and is just represented by a point in a network with a certain voltage. Hodgkin-Huxley equations [31] are able to reproduce the physical shape if it is combined to other differential equations, representing the dynamic of ion channels, but the complexity of these coupled equations that form a chaotic system [19], makes the system quite difficult to simulate for huge networks. If ion channels dynamic is neglected, the simplest model of voltage is the Integrate-and-Fire model [26]. With such models, it has been possible on supercomputers to simulate a human-scale cerebellar network reaching about 68×10^9 neurons [49].

However there is another point of view, which might allow us to simulate such massive networks with simplified models. Indeed, one can use much more random models to reproduce the essential dynamics of the neurons: their firing pattern. The randomization of not only the connectivity graph but also the dynamics on the graph is making the model closer to the data at hand and explain to a certain extent their variability. The introduction of randomness is not new and has been done in many models including Hodgkin-Huxley [13] and Leaky Integrate-and-fire (LIF for short) [27].

Here we want to focus on particular random models - point processes [46] - which have a particular property: time asynchrony, that is the inability of the model to have two spikes that are produced exactly at the same time by two different neurons. This includes in particular Hawkes models and variants such as GLM, Wold processes, Galves-Löcherbach models, and even some random LIF models with random or soft threshold, all of them having been used to fit real data [5, 15, 36, 38, 40, 42, 46].

This property, which is well known in mathematics [3, 5], combined with graph sparsity lead us to propose a new algorithm in [30]. The computational complexity of this new algorithm has been computed. Thanks to time asynchrony and to the computational activity tracking of firing neurons [32], we have shown in particular that if the graph is sparse, the complexity cost of the computation of a new point in the system is linear in the number of neurons. However the memory burden was too high to reach networks of 10^8 neurons.

In a preliminary work [18] focusing on the mathematical aspects of mean-field limits of LIFs, we formalized a way to deal with this memory aspect without putting it into practice: the main point is to not keep in memory the whole network, but to regenerate it when need be. Recently, the same idea, under the name of procedural connectivity, has been applied with success on LIF models in [24]: using GPU-based parallel programming, and without time asynchrony, the authors have been able to simulate a network of 4×10^6 neurons and 24×10^9 synapses on a desktop GPU computer.

But, as we show in the present article, the gain of procedural connectivity is even huger when combined with time asynchrony. Indeed, classical parallel programming usually uses a discrete simulation time and computes for all neurons (or synapses) what happens at each time step in a parallel fashion (even if spikes can be communicated in between two time steps [49]). At each time step, each process corresponding to a different neuron has to wait for the calculations of all the other processes to know what needs to be updated before computing the next step. With time asynchrony, we can leverage discrete-event programming [4, 33, 41, 45] to track the whole system in time by jumps: from one spike in the network to another spike in the network. Since a very small percentage of a brain is firing during a given unit of time [28], the gain we have is tremendous in terms of computations. Thanks to the procedural connectivity, the memory needed to access for the computation of each new spike is also controlled. Hence, thanks to procedural connectivity combined with time asynchrony, we propose a new algorithm for time asynchronous models, running sequentially on a single processor, thus simulating a realistic network of 10^8 neurons for which computational complexity as well as memory costs can be controlled beforehand.

4.2 Results

4.2.1 Time synchrony for parallel simulation

Brain simulation of large networks is usually done in parallel based on simulation synchronization. Of course, this depends on both the mathematical model at hand and the simulation algorithm. However, for most models, differential equations are used to derive the time course of the membrane voltage for each individual neurons. These equations are (approximately) solved by usual discrete-time numerical schemes (*cf.* Figure 4.1a) [24].

In this kind of implementation, when one presynaptic neuron fires at a time t, *i.e.*, emits a spike (red dots in Figure 4.1a), the synaptic transmission to post-synaptic neurons is done at the next time step $t + \Delta t$ (orange dots in Figure 4.1a). Between two synaptic transmissions, the membrane potential of a neuron evolves independently of the other neurons and can be computed in parallel (green dots in Figure 4.1a). However, since one does not know when a spike will be emitted in the network in advance, the membrane potential of all the neurons are classically computed in a synchronous way to be able to eventually transmit spikes, at each time step Δt of the algorithm.

4.2.2 Time asynchrony for sequential simulation

As said in the introduction, point processes models of neuronal network may guarantee time asynchrony if they have a stochastic intensity. Such processes include Hawkes processes, GLM approaches, Wold processes or Galvès Löcherbach models in continuous time [15, 36, 38, 40, 46]. Most of these models have proved their efficiency in terms of goodness-of-fit with respect to real spike train data [5, 12, 36, 38, 40, 46]. Often, the intensity in these models can be informally interpreted as a function of the membrane voltage and for more evidence, we refer the reader to [23], where the spike train of a motor neuron has been shown to be adequately modeled by a point process whose stochastic intensity is a function of the membrane voltage.

These point processes models differ from classical LIF, mainly because the higher the intensity (or the membrane potential) of a given neuron is, the more likely it is that the neuron fires, but this is never for sure. In classical LIF models, the neurons fire when their membrane potential reaches a fixed threshold. Therefore it may happen that if a presynaptic neuron fires, and if the corresponding postsynaptic neurons have a potential close to the threshold, then all the postsynaptic neurons fire at the exact same time. This phenomenon can be massive [6, 8]: this corresponds to the mathematical phenomenon of blow-up, which happens for some mean-field limits of such models. In this case, no time asynchrony is possible but such phenomenon is completely unrealistic from a biological point of view. There are LIF models with random or soft threshold [18, 42] which might not have this problem and which may also satisfy time asynchrony.

In [30], we proposed a discrete-event algorithm to simulate point processes with stochastic intensities. This algorithm is based on the theory of local independence graph [10], which is the directed neuronal network in our present case.

The algorithm works as follows (see Figure 4.1b). The spike events happen in continuous time in the system (up to the numerical precision). Once a spike on a particular presynaptic neuron happens (red dots in Figure 4.1b), the postsynaptic neurons are updated (orange dots in Figure 4.1b). The presynaptic and the post synaptic neurons compute their intensities (assimilated to membrane potential) and forecast its evolution (green arrows in Figure 4.1b) if nothing in between occurs in the system. They are therefore able to forecast their potential next spike (gray dots in Figure 4.1b). The algorithm maintains a scheduler containing all potential next spikes on all neurons and decides that the next neuron to fire effectively is the one corresponding to the minimum of these potential next spikes. For more details, we refer to [30].

The gain comes from the fact that neurons that are not firing a lot, do not require a lot of computation either. In particular we do not have to update all neurons at each spike but only the pre and post synaptic neurons that are involved in the spiking event. This is the main difference with the parallel simulation framework detailed above. The other difference is that we can work with arbitrary precision, typically 10^{-15} if necessary, without impeding the time complexity of the algorithm.

Note that the whole algorithm is possible only because two neurons in the network will not spike at the same time: the whole concept is based on time asynchrony to be able to jump from one spike in the system to the next spike in the system. Of course, this is true only up to numerical precision: if two potential next spikes (gray dots on Figure 4.1b) happen at the exact same time with resolution 10^{-15}, by convention the neuron with the smaller index is said to fire. But the probability of such event is so small that this is not putting the simulation in jeopardy.

Also note that this does not prevent neurons to eventually synchronize frequently over a small time duration of a few milliseconds, as defined for instance in [47] and the references therein.

4.2.3 Procedural connectivity

One of the memory burden of both methods (parallel and time asynchrony) comes from the fact that a classic implementation stores the whole connectivity graph, which is huge for brain scale models.

If the connectivity is the result of a random graph and that each presynaptic neuron is randomly connected to its postsynaptic neurons, one can store the random seed instead of the result of the random attribution. Hence the whole graph is never stored in full but only regenerated when need be (see Figure 4.1b). The random connectivity is regenerated at each spike taking advantage

of the deterministic nature of the pseudo-random generator used in the simulation. Storing the generator initial seed, the seed of each neuron is computed based on initial seed value and neuron index (see Figure 4.1c). With this method, only the initial seed is stored in memory. Of course this dynamic regeneration at each spike has a cost in terms of time complexity, but this cost is negligible with respect to the other computations that need to be made and this saves memory.

This method has been evoked at first in [18] for time asynchronous algorithms, without being put into practice, whereas this method has been already implemented with success on parallel programming with GPU [24].

4.2.4 Computational and memory costs

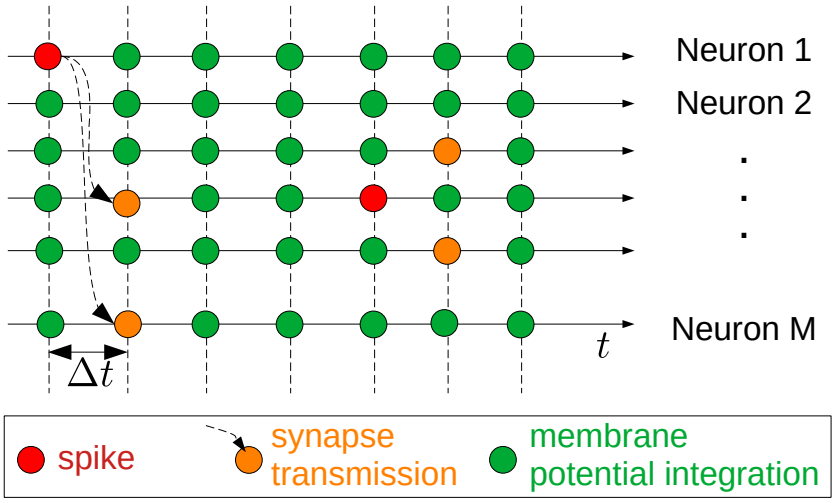
In [30], we obtained an accurate estimate of the complexity of the algorithm without procedural connectivity, for the simulation of linear Hawkes processes (*cf.* Equation 7 of [30]). This can be reused to compute the computational complexity of the same model, when the procedural connectivity step is added. Thus the overall time complexity of our algorithm is of the order of

$$\mathcal{O}\big(T\big[M d^2 \bar{m}^2 + \log(M) M d \bar{m} + M d \bar{m}\big]\big) \tag{4.1}$$

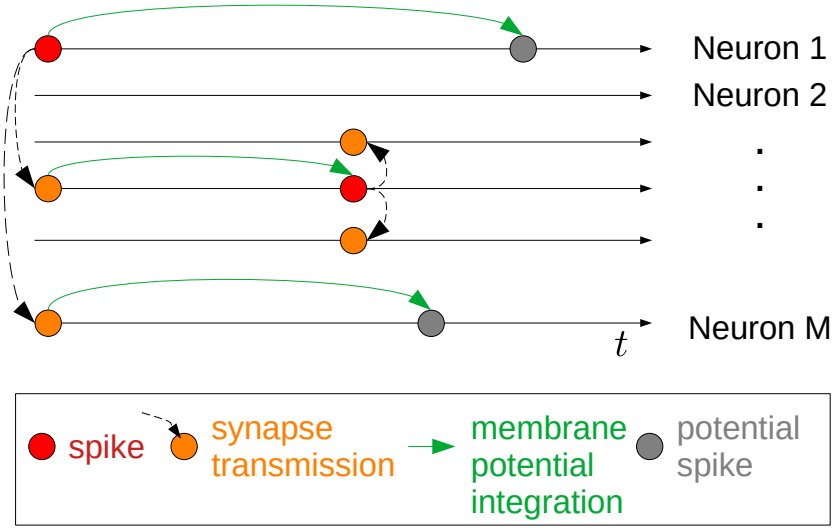
where M is the number of neurons, p the connection probability, $d = \lceil pM \rceil$ the average degree of the network, $\bar{m}$ the average firing rate of the network and T the simulation duration. The last term in (4.1) corresponds to the computational cost of the procedural connectivity at each spike, which is indeed negligible with respect to other terms, as explained before.

Note that such computational costs depend in particular on the intensity shape of the underlying point process model. The linearity of the Hawkes processes makes it easy to derive, whereas this can be much more cumbersome with other models such as stochastic LIF, which needs to compute the distribution of the time at which the threshold is reached.

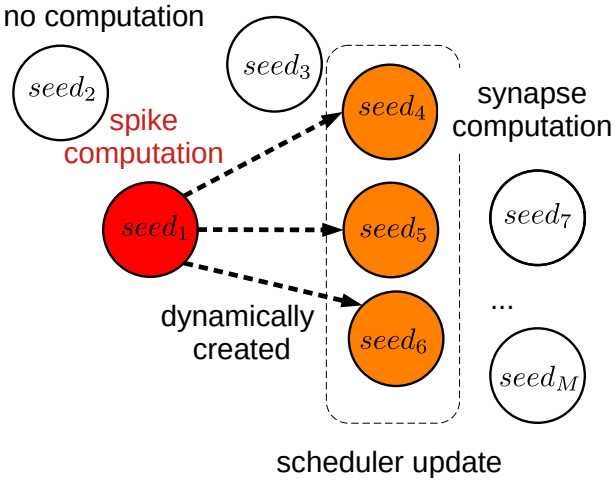
The maximal memory cost of the procedural connectivity, without the spike times storage, is of the order of $\mathcal{O}(d\omega) = \mathcal{O}(pM\omega)$, with ω the number of bits necessary for representing the index of a post-synaptic neuron.



(a) Discrete time approach for parallel simulation



(b) Discrete event approach for sequential simulation



(c) Time asynchrony and procedural connectivity

Figure 4.1: Neuronal computations in time and in the network

The memory cost of a static storage of the whole graph is of the order of $\mathcal{O}(dM\omega) = \mathcal{O}(pM^2\omega)$. We do not include in this the memory costs for the storage of each spike of each neuron. However, this cost is the same whatever the method and it is of the order of $MT\bar{m}\epsilon$, where ϵ is the number of bits necessary to represent a spike, which depends on the numerical precision with which time is recorded. If d is thought to be a fixed parameter, the memory cost of our complete algorithm with procedural connectivity is thus

$$\mathcal{O}(d\omega + MT\bar{m}\epsilon) \tag{4.2}$$

Conversely, the use of a static storage of the network is $\mathcal{O}(dM\omega + MT\bar{m}\epsilon)$. Note however that depending on what the program needs to return, we might not want to have the whole set of points but only summary statistics such as firing rates that will cost in memory much less than $\mathcal{O}(MT\bar{m}\epsilon)$.

4.2.5 Choice of brain scale parameters

Because of the precision of the actual measurements and the brain region and neuron variability, it is difficult to estimate quantitatively both physiological (number of synapses per neuron, etc.) and dynamic parameters (average firing rate, etc.) of neuronal networks in primates [22] and humans [21]. Only rough estimates are available. The human brain being the more computationally intensive, we estimate here its main parameters for simulation. Our goal is indeed to show the algorithm scalability to simulate large networks with such parameters.

To our knowledge, the best documented region of the human brain is the (neo)cortex. Based on the structural statistics (number of neurons and synaptic connections) of neuronal networks in the (neo)cortex, we extrapolate here their representative parameter values at brain scale.

The *firing rate of a neuron* in the brain can be estimated by the limited resources at its disposal, especially glucose. Measures of ATP consumption have shown (see [28]) that the firing rate of a neuron in human neocortex can be estimated around $0.16Hz$. Still based on ATP consumption, only 10% of the neurons in the neocortex can be active at the same time. So it seems coherent to choose an average of $0.16Hz$. These values can be extrapolated to the whole brain[1], as follows.

The neocortex represents 80% of the volume of the brain [44] and consumes 44% of its energy [28]. Considering that the energy consumed by the brain is proportional to the firing rate of the neurons, the power ratio then consists of

$$\frac{P_{cortex}}{P_{brain}} \sim \frac{V_{cortex}\bar{m}_{cortex}}{V_{brain}\bar{m}_{brain}},$$

[1] This calculus can be found on AI impact project webpage: https://aiimpacts.org/rate-of-neuron-firing/ (lastly verified: 02/09/2021)

Simulation duration	$T = 5s$
Mean firing rate	$\bar{m} = 0.3Hz$
Number of neurons simulated	$M = \{10^5, 10^6, 10^7, 10^8\}$
Synaptic connection probability	$p_M = 1000/M$
Postsynaptic duration	$\tau = 20ms$

Table 4.1: Neuronal network parameters at human brain scale level.

with $\bar{m}_{cortex}$ the mean firing rate of individual neurons in the neocortex (resp. in the brain) and V_{cortex} the volume of the neocortex (resp. in the brain). The *average firing rate of the brain* then consists of $\bar{m}_{brain} = 0.8 \times \frac{0.16}{0.44} = 0.29\ Hz$ per neuron.

This average firing rate should not be confused with the fact that particular neurons can have a much larger firing rate. Particularly, groups of neurons synchronize together for achieving a particular cognitive task: this is the concept of neuronal assemblies [17]. In an assembly, neurons can usually increase their rates to tens Hz (possibly $50Hz$) over a short duration. Therefore, we choose a *firing rate in the brain* where most of the neurons have a firing rate of $0.3Hz$ but some have a much higher firing rate (up to $50Hz$) using an heavy tailed distribution, see Materials and Methods and Table 4.2.

The average number of synaptic connections in human brains is hard to estimate and depends heavily on the neuron types and brain regions. For example, in the brain, it is assumed that the majority of neurons are cerebellum granule cells [43]. In [29], the number of synaptic connections to granule neurons is estimated to an average of only 4 connections, matching those observed anatomically. On the other hand, Purkinje neurons can have up to $200,000$ synapses on only one dendrite in the human brain [43]. The approximate number of synapses in the cortex is 0.6×10^{14} [11]. Assuming that the volume of the cortex represents around 80% of the volume of the brain, the number of synapses in the brain is of order 10^{14}. Considering that the number of neurons in the human brain is of order 10^{11} [21], we find that the average number of synapses is about $1,000$ synapses per neuron[2]. The *synaptic connection probability* thus depends on the number of neurons M: $p_M = \frac{1{,}000}{M}$.

Finally, an action potential arriving on one pre-synaptic neuron produces an Excitatory Post-Synaptic Potential (EPSP), or an Inhibitory PostSynaptic Potential (IPSP), in the postsynaptic neuron. The duration of these postsynaptic potentials is about $\tau = 20ms$ [43].

Therefore the parameters that we used in the simulation are indicated in Table 4.1. Notice that these parameters are generic and intuitive and can be taken easily into account in further studies, either at a biological or at theoretical model level.

[2]Calculus on AI impact project webpage: https://aiimpacts.org/scale-of-the-human-brain/ (lastly verified: 02/09/2021).

4.2.6 Software and hardware configurations

The simulations have been run on a Symmetric shared Memory multiProcessor (SMP) computer equipped with Intel CascadeLake@2.6GHz processors[3]. This kind of computer is used here to have access to larger memory capacities. At computational level, only one processor was used for the simulations. For small sizes of networks requiring small amounts of memory (*cf.* Figure 4.3b), *e.g.* a network of 10^6 neurons with a total of 10^9 synaptic connections, this computer is equivalent to a simple desktop computer. The simulation of such networks takes only 25 minutes for each biological second. This is of the order of the 4.13×10^6 neurons and 24.2×10^9 synapses simulated on GPUs [24], which takes about 15 minutes for each biological second. This GPU-based simulation was already running up to 35% faster than on 1024 supercomputer nodes (one rack of an IBM Blue Gene/Q) [16]. Our simulation only requires a single usual processor and no GPU.

The implementation of the algorithm is written in C++ (2011) programming language and compiled using g++ 9.3.0.

4.2.7 Firing rate at network level

Table 4.2 presents classical elementary statistics on the simulated firing rates, whereas Figure 4.2 presents the corresponding densities. As one can see in Section Material and Methods, the system is initialized with a lot of neurons whose spontaneous spiking activity is null. The system needs to warm up to have almost all neurons spiking. This explains why the density at $T = 5$s is still rippled whereas, at $T = 50s$, it looks much smoother. This last case corresponds basically to the stationary version of the process. Indeed, as explained in Section Material and Methods, the parameters of the Hawkes model (in particular the spontaneous spiking activity) have been fixed to achieve a certain stationary distribution of the firing rates (with mean 0.3Hz), which is heavy tailed to achieve records as large as 50 Hz. As one can see (even if at $T = 5$s the system is not warmed up yet with a lot of non spiking neurons), one can still achieve the desired average firing rate and extremal values. These basic statistics are not varying a lot with T (see Table 4.2). Note that the density plots are roughly the same for all configurations: with ripples at $T = 5$s and smooth curve at $T = 50$s.

Our approach is particularly adapted to simulate precisely and efficiently a huge disparity in frequency distributions. Indeed, our simulation algorithm [32] allows focusing efficiently the computing resources on highly spiking neurons without computing anything for almost silent

[3] We used v100l and v100xl partitions on Joliot-Curie supercomputer at TGCC as a Fenix Infrastructure resource. Each node of v100l and v100xl has Intel CascadeLake@2.6GHz processors. A node on v100l is a dual-socket one with 2x18 cores, each core having a memory of 10 GBytes, so the total amount of available memory is 360 GBytes. A node on v100xl is a quad-socket one with 4x18 cores, each core having a memory of 41.5 GBytes, so the total amount of available memory is 3 TBytes.

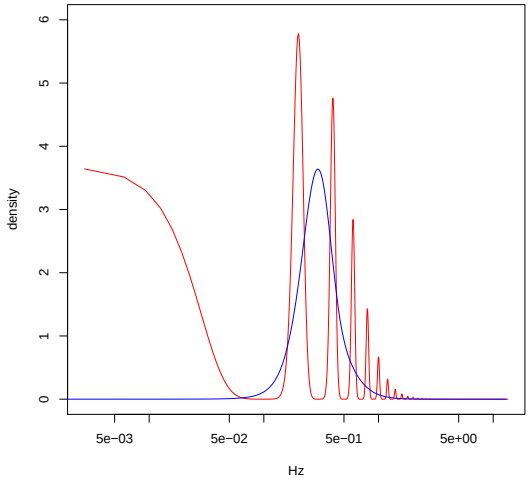


Figure 4.2: Densities (on a logarithmic scale) of the simulated firing rates in the network with $M = 10^6$ neurons and $d = 1000$ post-synaptic connections in average. In red, for $T = 5$s and in blue for $T = 50$s. These densities are obtained with a Gaussian kernel estimator with bandwidth 0.02.

neurons (*cf.* Figures 4.1a and 4.1b). The last advantage of our approach is to be able to store time stamps with a precision of $10^{-15}s$.

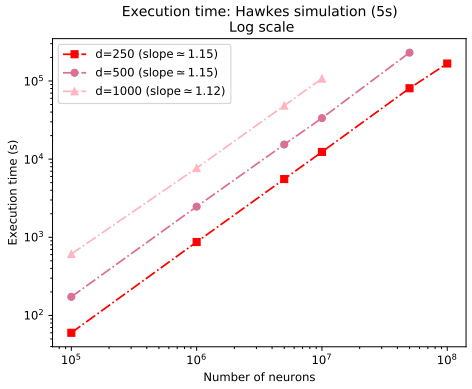
4.2.8 Execution times and memory usage

The simulation execution times are presented in Figure 4.3a for different sizes of neural networks and different numbers of synaptic connections (called children). The experimental execution times obtained are in agreement with (4.1) which predicts, for instance, $O(10^{12})$ operations for $M = 10^7$ and $d = 10^3$. The curves are almost linear (with slopes around 1.1) with respect to the number of neurons, for different numbers of synaptic connections.

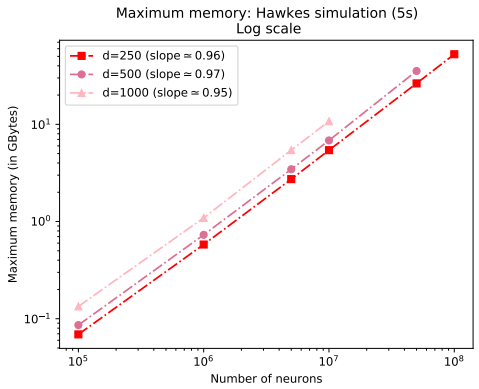
M	d	Average freq.	Freq. min.	Freq. max.	Freq. std.	Percentage of non spiking neuron
1e5	250	0.279 (0.279)	0 (0)	14 (13.94)	0.315 (0.222)	31.2 (0.01)
1e5	500	0.334 (0.333)	0 (0.02)	6.6 (5.76)	0.328 (0.218)	23.5 (0)
1e5	1000	0.399 (0.398)	0 (0.04)	10.4 (11.64)	0.345 (0.220)	16.9 (0)
1e6	250	0.267 (0.267)	0 (0)	19.2 (20.84)	0.308 (0.217)	33 (0.01)
1e6	500	0.322 (0.324)	0 (0)	38.4 (39.38)	0.329 (0.225)	25.1 (0.00)
1e6	1000	0.383 (0.387)	0 (0.02)	13.4 (12.9)	0.344 (0.223)	18.5 (0)
5e6	250	0.26 (0.261)	0 (0)	27.4 (28.5)	0.307 (0.217)	34.1 (0.02)
5e6	500	0.315 (0.316)	0 (0)	34.2 (34.1)	0.324 (0.220)	26 (0.00)
5e6	1000	0.377	0	19.8	0.342	19
1e7	250	0.258 (0.259)	0 (0)	23.2 (21.62)	0.306 (0.217)	34.4 (0.02)
1e7	500	0.311	0	21.6	0.322	26.4
1e7	1000	0.374	0	21.8	0.342	19.3
5e7	250	0.253	0	38.2	0.304	35.4
5e7	500	0.305	0	50.6	0.321	27.2
1e8	250	0.251	0	46.2	0.303	35.7

Table 4.2: Firing rates elementary statistics obtained by simulation for different sizes of neural networks and different numbers of synaptic connections and $T = 5s$. The number between parentheses displays the results at $T = 50s$ for the less complex simulations

.



(a) Simulation execution times for different sizes of networks.



(b) Memory usages for different sizes of networks.

Figure 4.3: Simulation execution times (4.3a) and memory usage (4.3b) for different sizes of networks.

The total amount of memory used is displayed in Figure 4.3b. They are in agreement with the procedural memory complexity of (4.2) and also almost linear (with slopes slightly below 1). Note in particular that for $M = 10^7$, $d = 10^3$, $\omega = 32$ and $\epsilon = 64$ (leading to a 10^{-15} precision

in time), the memory cost predicted by (4.2) is $O(10^{11})$ for the static implementation, whereas it is $O(10^9)$ for the procedural connectivity implementation. Besides notice that, as expected, increasing the average number of post-synaptic connections per neuron has few impact on the memory. Indeed, within the network, only the post-synaptic connections receiving spikes are dynamically generated.

4.3 Material and Methods

4.3.1 Model

For a set of M neurons, we first design the graph of interaction by saying that neuron j influences neuron i if a Bernoulli variable $Z_{j\to i}$ of parameter p is non zero. The resulting network is an Erdös-Rényii graph.

Once the network is fixed, we design the spike apparition thanks to a Hawkes process, that is a point process whose intensity is given by

$$\lambda^i(t) = \nu_i + \sum_{j=1}^{M} \int_0^t h_{j\to i}(t-\tau) dN_\tau^j,$$

with dN^j the point measure associated to neuron j. In this formula, ν_i represents the spontaneous firing rate of the neuron i if the other neurons do not fire, whereas $h_{j\to i}$ is the interaction function, that is $h_{j\to i}(u)$ is the increase (if positive) or decrease (if negative) that the firing rate of neuron i suffers due to a spike on j, which happens u seconds before.

We are interested in a particular case of the Hawkes process where all the interaction functions are always the same when they are non null. More precisely, we set the interaction function

$$h_{j\to i} = Z_{j\to i}\theta h,$$

where h is a fixed positive interaction function of integral 1, θ is a tuning parameter that we need to calibrate to avoid explosion of the process. We also set $h_{i\to i} = 0$ (no self interaction). We take $h = 50\mathbb{1}_{[0,0.02]}$: the interaction function is a constant and non zero only on a small interval of length 20ms, which corresponds to typical Post Synaptic Potentials in the brain.

Let us denote $H_{j\to i} = \int_0^{+\infty} h_{j\to i}(t)dt)$ and $H = (H_{j\to i})_{i,j=1,\dots,M}$ is the corresponding matrix (line i corresponds to a triggered neuron, column j to a triggering neuron).

Note in particular that in this model, there are only excitatory neurons : if in the brain, there are inhibitory neurons, this will only reduce the number of points without changing the complexity.

Moreover when all the interaction functions are non negative, we can easily understand the explosion condition.

Indeed, this Hawkes process explodes, that is, it produces an exponentially increasing number of point per unit of time (see [9]) if the spectral radius of H is larger than 1.

If (*Condition Stat*) the spectral radius is strictly smaller than 1 [20], then a stationary version exists and the corresponding vector of mean firing rates $m = (m_i)_{i=1,...,M}$ is given by

$$m = (I - H)^{-1} \nu. \tag{4.3}$$

Note also that if we start the simulation without points before 0 in (*Condition Stat*), the process is not stricto sensu stationary but it will converge to an equilibrium given by the stationary state (ergodic theorem) and that the number of points that will be produced is always smaller than the stationary version.

In the present case we want (i) to prevent explosion and (ii) to reach a certain vector m which is biologically realistic (average around 0.3 Hz, records around 50 Hz). Both of these calibrations can be done mathematically beforehand in the Hawkes model : we can guarantee the behavior of the whole system even before performing the simulation, whereas this might be much more intricate for other models such as LIF.

4.3.2 Choice of theta or how to avoid explosion

Note that $H = \theta \mathcal{Z}$, with $\mathcal{Z} = (Z_{j\to i})_{i,j=1,...,M}$. So if we can compute the largest eigenvalue of $\mathcal{Z}$ or an upper bound, we can decide how to choose θ.

We can use Gershgorin circles [48] to say that any complex eigenvalue λ of $\mathcal{Z}$ satisfies (because the diagonal is null),

$$|\lambda| \leq \max_{i=1,...,M} \sum_{j\neq i} Z_{j\to i}.$$

Therefore the spectral radius is upper bounded by $\max_{i=1,...,M} B_i$, where $B_i = \sum_{j\neq i} Z_{j\to i}$. This random quantity can be computed for small networks but it is clearly too intensive in our setting: indeed, with the procedural connectivity implementation, it is always easy to access the children ℓ of a given i, i.e. such that $i \to \ell$ is in the graph, but we need to look at all the neurons in the graphs to find out the set of parents j of i, i.e. such that $j \to i$ is in the graph. However, probabilistic estimates might be computed mathematically. Indeed B_i is just a sum of i.i.d. Bernoulli variables. So we can apply Bernstein's inequality [2]. This leads to, for all positive x,

$$\begin{aligned}
&\forall i = 1, ..., M, \\
&\mathbb{P}(B_i \geq (M-1)p + \sqrt{2(M-1)p(1-p)x} + x/3) \leq e^{-x},
\end{aligned}$$

and, by union bound, for the maximum

$$\mathbb{P}(\max_{i=1,\dots,M} B_i \geq (M-1)p + \sqrt{2(M-1)p(1-p)x} + x/3) \leq Me^{-x}.$$

Therefore let us fix a level α, say 1%, and take $x = \log(M) + \log(1/\alpha)$ in the previous equation. We obtain that with probability larger than $1-\alpha$, the spectral radius of $\mathscr{L}$ is upper bounded

$$\rho_{\max} = (M-1)p + \xi_\alpha$$

with

$$\xi_\alpha = \sqrt{2(M-1)p(1-p)[\log(M) + \log(1/\alpha)]} + [\log(M) + \log(1/\alpha)]/3$$

Note that $\rho_{\max}$ is roughly $(M-1)p$, which is the largest eigenvalue of $\mathbb{E}(\mathscr{L})$. Finally it means that if we take $\theta < 1/\rho_{\max}$, the process will not explode with probability larger than $1-\alpha$. In practice, to ensure a strong enough interaction, we take $\theta = 0.9\rho_{max}^{-1}$.

4.3.3 Choice of nu_i or how to constraint the distribution of the mean firing rates

The first step consists in deciding for a target distribution for the m_i. We have chosen to pick the m_i's independently as $0.1X$ where X is the absolute value of a student variable with mean 3 and 4 degrees of freedom. The choice of the student variable was driven by the wish of having a moderate heavy tail, which will ensure records around 50 Hz and a mean around 0.3Hz. The problem is that the m_i's are not parameters of the model, so we need to understand how to tune ν_i to get such m_i's. Note that by inverting (4.3), we get that

$$(I - H)m = \nu$$

that is for all i

$$\nu_i = m_i - \theta \sum_{j \neq i} m_j Z_{j \to i},$$

which is very intuitive [39]. Indeed the spontaneous rate that we need to put is the mean firing rate m_i minus what can be explained with the parents of i.

So in theory, the Hawkes model is very easy to tune for prescribed firing rates since there is a linear relationship between both. However, and for the same reasons as before, it might be too computationally intensive to compute this explicitly.

One possible way is to again use concentration inequalities, but this time on $\sum_{j\neq i} m_j Z_{j\to i}$ and not on B_i. However we decided to do something simpler, which works well (as seen in Figure 4.2).

Indeed $\sum_{j\neq i} m_j Z_{j\to i}$ is a sum of about $(M-1)p \simeq 1000$ i.i.d variables with mean $\bar{m} = 0.3$Hz. Hence it should be close to $\bar{m}B_i$. With the previous computations, we know already that ν_i should therefore be larger than $m_i - \theta\rho_{\max}\bar{m}$.

With the previous choice of $\theta = 0.9\rho_{\max}^{-1}$, we have chosen to take the positive part for the ν_i's in the simulation, that is :

$$\nu_i = \max(m_i - 0.9\bar{m}, 0).$$

Therefore ν_i remains positive or null, which guarantees that the Hawkes process stays linear. However, this also means that a non negligible portion of the neurons start with a null spontaneous firing rate, which explains the ripples of Figure 4.2.

With this choice, we cannot hope to have exactly the same distribution as the desired m_i's, but it conserves the same heavy tail and roughly the same mean firing rate as the one we wanted, as one can see on Table 4.2.

4.4 Discussion

Thanks to time asynchrony, we propose a new scalable algorithm to the simulate spiking activity of neuronal networks. We are able to generate roughly the same firing pattern as a real brain for a range between 10^5 and 10^8 neurons, in a few minutes on a single processor, most parameters being tuned thanks to general considerations inferred from the literature. Corresponding computational and memory complexities are shown to be both linear.

At simulation level, whereas usual simulations are based on the continuous variation of the electrical potentials of LIF neurons, point processes lead to much more efficient simulations. In particular, instead of computing the small continuous variations for all neurons, only discrete spikes and their interactions are simulated in the network. Between two spikes no computations are done. Point processes also lead to time asynchrony (two spikes cannot occur at the same time in the network), which is a fundamental hypothesis for the algorithm to work.

Combining the time asynchrony hypothesis with procedural connectivity drastically reduces the memory consumption and also, for the same network activity, reduces the computations per spikes (*cf.* Figures 4.1a and 4.1b). In particular, complexities (both theoretical and concrete) can be computed and proved to be almost linear in the number of neurons, when Hawkes processes are generated, leading to simulation scalability of the whole approach without precision loss.

Both modeling and simulation results open many research perspectives. We are currently developing new discrete event algorithms that are able to simulate the spiking activity of neurons

embedded in potentially infinite neuronal networks [35]. This paves the path for simulation of parts of the brain as an open physical system.

Furthermore, the minimal number of computations and memory storage obtained here open new exciting perspectives with respect to massive neuromorphic computers, by improving the energy saving consumption of neuromorphic components [37].

Finally, if we have proved that the simulation is doable, the point process model used here can be calibrated further on real data, by incorporating inhibition and more variability in the interaction functions. Also, this model can be used for reconstructing the functional connectivity of experimentally recorded neurons [39, 40] to have access to more realistic interaction functions. Besides, as only few computing resources (one single processor) is used with a minimal memory amount, this opens new possibilities to run in parallel many independent replications of stochastic simulations of large networks. This is particularly interesting for calibrating models based on real data collections.

ACKNOWLEDGMENTS. This work is part of the project HyperBrain from Human Brain Project (HBP) EBRAINS EU initiative. The simulations were run on Fenix Infrastructure resources, which are partially funded from the European Union's Horizon 2020 research and innovation program through the ICEI project under the grant agreement No. 800858. Our research was supported by the French government, through CNRS, the UCA^{Jedi} and 3IA Côte d'Azur Investissements d'Avenir managed by the National Research Agency (ANR-15- IDEX-01 and ANR-19-P3IA-0002), directly by the ANR project ChaMaNe (ANR-19-CE40-0024-02) and by the interdisciplinary Institute for Modeling in Neuroscience and Cognition (NeuroMod) of the Université Côte d'Azur.

References for Chapter 4

1. K. Amunts, A. C. Knoll, T. Lippert, C. M. Pennartz, P. Ryvlin, A. Destexhe, V. K. Jirsa, E. D'Angelo, and J. G. Bjaalie. "The Human Brain Project—Synergy between neuroscience, computing, informatics, and brain-inspired technologies". *PLoS biology* 17:7, 2019, e3000344.
2. S. Boucheron, G. Lugosi, and P. Massart. *Concentration inequalities*. A nonasymptotic theory of independence, With a foreword by Michel Ledoux. Oxford University Press, Oxford, 2013.
3. P. Brémaud. *Point processes and queues*. Martingale dynamics, Springer Series in Statistics. Springer-Verlag, New York-Berlin, 1981.

4. R. Brette, M. Rudolph, T. Carnevale, M. Hines, D. Beeman, J. M. Bower, M. Diesmann, A. Morrison, P. H. Goodman, F. C. Harris, et al. "Simulation of networks of spiking neurons: a review of tools and strategies". *Journal of computational neuroscience* 23:3, 2007, pp. 349–398.

5. E. N. Brown, R. Barbieri, V. Ventura, R. E. Kass, and L. Frank. "The Time-Rescaling Theorem and Its Application to Neural Spike Train Data Analysis". *Neural Computation* 14:2, 2002, pp. 325–346.

6. M. J. Cáceres, J. A. Carrillo, and B. Perthame. "Analysis of nonlinear noisy integrate & fire neuron models: blow-up and steady states". *J. Math. Neurosci.* 1, 2011, Art. 7, 33.

7. M. R. Dando. *Japan's Brain/MINDS Project*. 2020.

8. F. Delarue, J. Inglis, S. Rubenthaler, and E. Tanré. "Global solvability of a networked integrate-and-fire model of McKean-Vlasov type". *Ann. Appl. Probab.* 25:4, 2015, pp. 2096–2133.

9. S. Delattre, N. Fournier, and M. Hoffmann. "Hawkes processes on large networks". *Ann. App. Probab.* 26, 2016, pp. 216–261.

10. V. Didelez. "Graphical models of markes point processes based on local independence". *J.R. Statist. Soc. B* 70:1, 2008, pp. 245–264.

11. D. A. Drachman. "Do we have brain to spare?" *Neurology* 64:12, 2005, pp. 2004–2005. ISSN: 0028-3878. DOI: 10.1212/01.WNL.0000166914.38327.BB. eprint: https://n.neurology.org/content/64/12/2004.full.pdf. URL: https://n.neurology.org/content/64/12/2004.

12. A. Duarte, A. Galves, E. Löcherbach, and G. Ost. "Estimating the interaction graph of stochastic neural dynamics". *Bernoulli* 25:1, 2019, pp. 771–792.

13. R. F. Fox. "Stochastic versions of the Hodgkin-Huxley equations". *Biophysical journal* 72:5, 1997, pp. 2068–2074.

14. E. Gal, M. London, A. Globerson, S. Ramaswamy, M. W. Reimann, E. Muller, H. Markram, and I. Segev. "Rich cell-type-specific network topology in neocortical microcircuitry". *Nature Neuroscience* 20:7, 2017, pp. 1004–1013. ISSN: 1546-1726. DOI: 10.1038/nn.4576. URL: https://doi.org/10.1038/nn.4576.

15. A. Galves and E. Löcherbach. "Infinite Systems of Interacting Chains with Memory of Variable Length—A Stochastic Model for Biological Neural Nets". *Journal of Statistical Physics* 151:5, 2013, pp. 896–921. URL: https://doi.org/10.1007/s10955-013-0733-9.

16. M.-O. Gewaltig and M. Diesmann. “Nest (neural simulation tool)”. *Scholarpedia* 2:4, 2007, p. 1430.

17. F. Grammont and A. Riehle. “Spike synchronization and firing rate in a population of motor cortical neurons in relation to movement direction and reaction time”. *Biological cybernetics* 88:5, 2003, pp. 360–373.

18. Grazieschi, Paolo, Leocata, Marta, Mascart, Cyrille, Chevallier, Julien, Delarue, François, and Tanré, Etienne. “Network of interacting neurons with random synaptic weights”. *ESAIM: ProcS* 65, 2019, pp. 445–475. DOI: 10.1051/proc/201965445. URL: https://doi.org/10.1051/proc/201965445.

19. J. Guckenheimer and R. A. Oliva. “Chaos in the Hodgkin–Huxley model”. *SIAM Journal on Applied Dynamical Systems* 1:1, 2002, pp. 105–114.

20. A. G. Hawkes and D. Oakes. “A cluster process representation of a self-exciting process”. *J. Appl. Probability* 11, 1974, pp. 493–503.

21. S. Herculano-Houzel and R. Lent. “Isotropic Fractionator: A Simple, Rapid Method for the Quantification of Total Cell and Neuron Numbers in the Brain”. *Journal of Neuroscience* 25:10, 2005, pp. 2518–2521. ISSN: 0270-6474. DOI: 10.1523/JNEUROSCI.4526-04.2005. eprint: https://www.jneurosci.org/content/25/10/2518.full.pdf. URL: https://www.jneurosci.org/content/25/10/2518.

22. S. Herculano-Houzel, C. E. Collins, P. Wong, and J. H. Kaas. “Cellular scaling rules for primate brains”. *Proceedings of the National Academy of Sciences* 104:9, 2007, pp. 3562–3567. ISSN: 0027-8424. DOI: 10.1073/pnas.0611396104. eprint: https://www.pnas.org/content/104/9/3562.full.pdf. URL: https://www.pnas.org/content/104/9/3562.

23. P. Jahn, R. W. Berg, J. Hounsgaard, and S. Ditlevsen. “Motoneuron membrane potentials follow a time inhomogeneous jump diffusion process”. *J. Comput. Neurosci.* 31:3, 2011, pp. 563–579.

24. J. C. Knight and T. Nowotny. “Larger GPU-accelerated brain simulations with procedural connectivity”. *Nature Computational Science* 1, 2021, pp. 136–142.

25. W. Koroshetz, J. Gordon, A. Adams, A. Beckel-Mitchener, J. Churchill, G. Farber, M. Freund, J. Gnadt, N. S. Hsu, N. Langhals, et al. “The state of the NIH BRAIN initiative”. *Journal of Neuroscience* 38:29, 2018, pp. 6427–6438.

26. L. Lapicque. “Recherches quantitatives sur l’excitation electrique des nerfs traitee comme une polarization.” *Journal of Physiology and Patholologyy* 9, 1907, pp. 620–635.

27. A. A. Lazar and E. A. Pnevmatikakis. "Reconstruction of sensory stimuli encoded with integrate-and-fire neurons with random thresholds". *EURASIP Journal on Advances in Signal Processing* 2009, 2009, pp. 1–14.

28. P. Lennie. "The Cost of Cortical Computation". *Current Biology* 13:6, 2003, pp. 493–497. ISSN: 0960-9822. DOI: https://doi.org/10.1016/S0960-9822(03)00135-0. URL: http://www.sciencedirect.com/science/article/pii/S0960982203001350.

29. A. Litwin-Kumar, K. D. Harris, R. Axel, H. Sompolinsky, and L. Abbott. "Optimal Degrees of Synaptic Connectivity". *Neuron* 93:5, 2017, 1153–1164.e7. ISSN: 0896-6273. DOI: https://doi.org/10.1016/j.neuron.2017.01.030. URL: https://www.sciencedirect.com/science/article/pii/S0896627317300545.

30. C. Mascart, A. Muzy, and P. Reynaud-bouret. *Efficient Simulation of Sparse Graphs of Point Processes*. 2020. arXiv: 2001.01702 [stat.CO].

31. D. A. McCormick, Y. Shu, and Y. Yu. "Hodgkin and Huxley model—still standing?" *Nature* 445:7123, 2007, E1–E2.

32. A. Muzy. "Exploiting Activity for the Modeling and Simulation of Dynamics and Learning Processes in Hierarchical (Neurocognitive) Systems". *Computing in Science Engineering* 21:1, 2019, pp. 84–93. ISSN: 1558-366X. DOI: 10.1109/MCSE.2018.2889235.

33. A. Muzy, B. P. Zeigler, and F. Grammont. "Iterative specification as a modeling and simulation formalism for I/O general systems". *IEEE Systems Journal* 12:3, 2017, pp. 2982–2993.

34. R. R. Netz and W. A. Eaton. "Estimating computational limits on theoretical descriptions of biological cells". *PNAS* 118:6, 2021.

35. T. C. Phi, A. Muzy, and P. Reynaud-Bouret. "Event-Scheduling Algorithms with Kalikow Decomposition for Simulating Potentially Infinite Neuronal Networks". *SN Computer Science* 1:1, 2020. DOI: 10.1007/s42979-019-0039-3. URL: https://hal.archives-ouvertes.fr/hal-02321497.

36. J. Pillow, J. Shlens, L. Paninski, A. Sher, E. Chichilnisky, and E. Simoncelli. "Spatio-temporal correlations and visual signalling in a complete neuronal population". *Nature* 454, 2008, pp. 995–999.

37. L. A. Plana, J. Garside, J. Heathcote, J. Pepper, S. Temple, S. Davidson, M. Luján, and S. Furber. "spiNNlink: FPGA-Based Interconnect for the Million-Core SpiNNaker System". *IEEE Access* 8, 2020, pp. 84918–84928.

38. C. Pouzat and A. Chaffiol. "Automatic spike train analysis and report generation. An implementation with R, R2HTML and STAR". *Journal of Neuroscience Methods* 181:1, 2009, pp. 119–144. ISSN: 0165-0270. DOI: https://doi.org/10.1016/j.jneumeth.2009.01.037. URL: https://www.sciencedirect.com/science/article/pii/S0165027009001058.

39. P. Reynaud-Bouret, A. Muzy, and I. Bethus. "Towards a mathematical definition of functional connectivity". 2021. URL: https://hal.archives-ouvertes.fr/hal-03093516.

40. P. Reynaud-Bouret, V. Rivoirard, F. Grammont, and C. Tuleau-Malot. "Goodness-of-Fit Tests and Nonparametric Adaptive Estimation for Spike Train Analysis". *The Journal of Mathematical Neuroscience* 4:1, 2014, p. 3.

41. M. Rudolph and A. Destexhe. "Analytical integrate-and-fire neuron models with conductance-based dynamics for event-driven simulation strategies". *Neural computation* 18:9, 2006, pp. 2146–2210.

42. L. Sacerdote and M. T. Giraudo. "Stochastic Biomathematical Models". In: vol. 2058. Lecture Notes in Mathematics, Springer, 2013. Chap. Stochastic Integrate and Fire Models: A Review on Mathematical Methods and Their Applications, pp. 99–148.

43. G. M. Shepherd. *The synaptic organization of the brain*. Oxford university press, 2004.

44. H. Stephan, H. Frahm, and G. Baron. "New and revised data on volumes of brain structures in insectivores and primates". *Folia primatologica* 35:1, 1981, pp. 1–29.

45. J. D. Touboul and O. D. Faugeras. "A Markovian event-based framework for stochastic spiking neural networks". *Journal of computational neuroscience* 31:3, 2011, pp. 485–507.

46. W. Truccolo, U. T. Eden, M. R. Fellows, J. P. Donoghue, and E. N. Brown. "A Point Process Framework for Relating Neural Spiking Activity to Spiking History, Neural Ensemble, and Extrinsic Covariate Effects". *Journal of Neurophysiology* 93:2, 2005, pp. 1074–1089.

47. C. Tuleau-Malot, A. Rouis, F. Grammont, and P. Reynaud-Bouret. "Multiple Tests Based on a Gaussian Approximation of the Unitary Events Method with Delayed Coincidence Count". *Neural Computation* 26:7, 2014, pp. 1408–1454. ISSN: 0899-7667. DOI: 10.1162/NECO_a_00604. URL: https://doi.org/10.1162/NECO_a_00604 (visited on 06/06/2021).

48. R. Varga. *Gershgorin and his circles*. Springer Series in Computational Mathematics. Springer-Verlag, 2004.

49. H. Yamaura, J. Igarashi, and T. Yamazaki. "Simulation of a human-scale cerebellar network model on the k computer". *Frontiers in neuroinformatics* 14, 2020, p. 16.

5 Conclusion and perspectives

Point processes, and in a certain way stochastic processes, model sporadic events in time. Simulation algorithms are well known but are often too slow for large-scale applications such as brain-scale simulation. On the other hand, discrete-event system specification is a formalism which deal with the modeling and simulation of system changing state at the occurrence of events. It has been used and improved over the years for modeling and simulating very complex systems efficiently.

While there is a large corpus of research on improving simulation algorithms for point processes and stochastic processes, as far as we know, little work has been done to combine the known simulation algorithms for point processes and stochastic processes with discrete-event techniques. This is surprising because point process models are, at their core, discrete-event systems, undergoing state change only at the time when a new point is generated.

In this thesis we have worked on synthesizing the classical algorithms for simulation of point and stochastic processes with the methods and formalism from the field of discrete-event systems. Our first approach was to formalize the Hawkes process model in the terms of the DEVS formalism. We also simplified the architecture of our home-made DEVS simulator, GODDESS, to get a prototype for future work on point process simulation using discrete-event techniques.

A second initiative was launched with the 2017 edition of the CEMRACS summer school, during which we successfully applied the discrete-event formalism to the modeling and simulation of the Ornstein-Uhlenbeck model of spiking neurons. The concept of procedural connectivity was also rediscovered and applied for the first time during this particular project. This project helped strengthening the idea that the DEVS formalism and using techniques from the field of discrete-event simulation was indeed a good idea for bettering current simulation algorithms for both point and stochastic processes.

During the third phase of this thesis, we focused on re-designing and re-implementing the point process simulator in C++. We called the simulator SPIKES, in reference to the spikes that we model using point processes. SPIKES uses both the new algorithm that we proposed for the simulation of point processes, and the procedural connectivity introduced for stochastic process simulation to lower the memory complexity of both the algorithm and the data structures, compared to classical algorithms and implementations. The simulator currently works on single processor, yet it is still

able to achieve real-time simulation of networks of thousands of processes/neurons. The simulator has proven able to simulate networks of processes at the same scale as the number of neurons in a small monkey brain.

An interesting fact that was discovered at the end of this thesis is that the concept of procedural connectivity has already been introduced by Izhikevich in 2005, in an unpublished work. His personal website references the unpublished experimentation [60]. Izhikevich explains he tried to simulate a large network of 100 billions spiking neurons, which is the same order of magnitude as the number of neurons in the human brain. What is interesting is the explanations Izhikevich gives to this simulation [61]: he tells that 1 second of simulation took about 50 days on a 27 3GHz cluster (in 2005), and that according to Moore's law, the simulation of an entire brain would be possibly achieved by 2016, on a cluster of 911250 processors each running at 384GHz. The hope was clearly that even more massive parallelization than what was available at that time would solve the simulation limits, and yet in 2020 no satisfying full-scale brain simulation has been run.

It is worthy of noting that Izhikevich achieved his impressive technical feat by using a similar technique for the storage of the synapses as we developed in chapters 2 and 4. Indeed, he implemented the model so as to regenerate the anatomy at every time step (1ms). There is no further explanation as whether it was the anatomy of the whole brain that was regenerated at every time step, or just a subgraph. Indeed, some of the ideas that made this thesis work possible had been suggested, at least in parts, 15 years before this manuscript came to life. There is no more humbling sentiment than to realize how much we step on the shoulders of giants.

A lot of work has been done for improving the parallelization of current algorithms for neuronal simulation on computer clusters [53, 131, 137] and with the use of GPU [31, 70, 105, 108]. Yet, we would like to defend the single processor approach in this conclusion.

Resource-savvy algorithms will become more and more important in an energy-stressed world, while allowing neuroscience researchers to work at the scale of a human brain without the need to use the resources of expensive supercomputers. The idea of promoting the research for more resource-savvy algorithms is on par with the resource-effectiveness of the brain compared to the simulations: the current consensus is that the brain takes 20% of the energy of the whole body [24, 50, 90, 114, 146], which implies an energy consumption of $20 - 30$W: assuming 2000 kCal/day consumed by an adult, this would translate to about 400 kCal/day consumed by the brain, or less than 20W. A higher estimation is given in [50]: 600kCal/day, or 30W, is estimated for 100 billion neurons, and a quick computation gives about 25W for 86 billion neurons. The Japanese K computer, which held the 20th place in the Top500 ranking of supercomputers in 2019 when it was decommissioned, can simulate a human-scale cerebellar model with 68 billion neurons [149], but at the cost of 12MW [141]. By comparison, a generic laptop computer consumes less than 50W [10].

Moreover, the DEVS formalism can help the development of models and simulation software suitable for both single processor and parallel computing. The DEVS formalism can easily be adapted in a parallel version [23, 142, 154], and is able to handle both discrete-event and discrete-time models [142, 154]. The advantage of using DEVS as a building block for designing models and simulation software is that it opens the possibility of simulating a large range of models, even when they use different formalisms [57, 154]. Finally, theoretical analysis by Zeigler [153] has shown that the Parallel DEVS protocol is very close to optimality in terms of performance, so using the DEVS formalism could help building efficient yet generic simulation software. This could also be a path towards the integration of more interdisciplinary work at the confluence of mathematics, computer science and neuroscience for example.

The research presented in this thesis is already coming to fruition. With the ability to simulate greater network of point and stochastic processes, it is possible to verify the performance of old and new statistical tools. For instance, numerous techniques for reconstructing the functional connectivity of a biological neural network from the measured spiking activity of the neurons have been proposed in the past [3, 22, 27, 123]. We worked on improving the algorithlms for the computation of the functional connectivity using the LASSO method, as proposed in [80], with encouraging results thus far.

In addition, a collaboration with the SED team at the Inria laboratory of Sophia-Antipolis, a graphical interface for the SPIKES simulator and the reconstruction algorithm is being implemented. The goal is to couple the simulation and reconstruction software to allow neuroscientist to be able to quickly and easily transform raw spiking data into a reconstructed neural network, which can then be verified using the simulation software.

Finally, a newly proposed algorithm by Phi et al. [117] allow the simulation of neurons in a potentially infinite network of neurons. This could even improve the performance of the simulation of multivariate Hawkes models of neural networks. The idea would be to use the same kind of discrete-event techniques that were used in this thesis to improve the result of [117].

Bibliography

1. L. F. Abbott. "Lapicque's introduction of the integrate-and-fire model neuron (1907)". *Brain research bulletin* 50:5-6, 1999, pp. 303–304.
2. A. Abraham. "Artificial neural networks". *Handbook of measuring system design*, 2005.
3. I. M. de Abril, J. Yoshimoto, and K. Doya. "Connectivity inference from neural recording data: Challenges, mathematical bases and research directions". *Neural Networks* 102, 2018, pp. 120–137.
4. E. Bacry, I. Mastromatteo, and J.-F. Muzy. "Hawkes processes in finance". *Market Microstructure and Liquidity* 1:01, 2015, p. 1550005.
5. A. Baddeley, P. Gregori, J. Mateu, R. Stoica, and D. Stoyan. *Case studies in spatial point process modeling*. Vol. 185. Springer, 2006.
6. O. Balci. "The implementation of four conceptual frameworks for simulation modeling in high-level languages". In: *Proceedings of the 20th conference on Winter simulation*. 1988, pp. 287–295.
7. N. M. Bannon, M. Chistiakova, and M. Volgushev. "Synaptic plasticity in cortical inhibitory neurons: what mechanisms may help to balance synaptic weight changes?" *Frontiers in Cellular Neuroscience* 14, 2020.
8. L. Bauwens and N. Hautsch. "Modelling financial high frequency data using point processes". In: *Handbook of financial time series*. Springer, 2009, pp. 953–979.
9. B. P. Bean. "The action potential in mammalian central neurons". *Nature Reviews Neuroscience* 8:6, 2007, pp. 451–465.
10. G. Bekaroo and A. Santokhee. "Power consumption of the Raspberry Pi: A comparative analysis". In: *2016 IEEE International Conference on Emerging Technologies and Innovative Business Practices for the Transformation of Societies (EmergiTech)*. 2016, pp. 361–366. DOI: 10.1109/EmergiTech.2016.7737367.
11. P. Black. *Dictionary of Algorithms and Data Structures*. en. 1998.

12. A. Bouchard-Côté, S. J. Vollmer, and A. Doucet. "The bouncy particle sampler: A non-reversible rejection-free Markov chain Monte Carlo method". *Journal of the American Statistical Association* 113:522, 2018, pp. 855–867.

13. P. Brémaud. *Point processes and queues: martingale dynamics*. Vol. 50. Springer, 1981, pp. 1–13, 255–295.

14. P. Brémaud and L. Massoulié. "Stability of nonlinear Hawkes processes". *The Annals of Probability* 24:3, 1996, pp. 1563–1588. DOI: 10.1214/aop/1065725193. URL: https://doi.org/10.1214/aop/1065725193.

15. R. Brette. "Philosophy of the Spike: Rate-Based vs. Spike-Based Theories of the Brain". *Frontiers in Systems Neuroscience* 9, 2015, p. 151. ISSN: 1662-5137. DOI: 10.3389/fnsys.2015.00151. URL: https://www.frontiersin.org/article/10.3389/fnsys.2015.00151.

16. R. Brette, M. Rudolph, T. Carnevale, M. Hines, D. Beeman, J. M. Bower, M. Diesmann, A. Morrison, P. H. Goodman, F. C. Harris, et al. "Simulation of networks of spiking neurons: a review of tools and strategies". *Journal of computational neuroscience* 23:3, 2007, pp. 349–398.

17. P. Brodal. *The central nervous system: structure and function*. oxford university Press, 2004.

18. E. N. Brown, R. Barbieri, V. Ventura, R. E. Kass, and L. M. Frank. "The time-rescaling theorem and its application to neural spike train data analysis". *Neural computation* 14:2, 2002, pp. 325–346.

19. A. N. Burkitt. "A review of the integrate-and-fire neuron model: I. Homogeneous synaptic input". *Biological cybernetics* 95:1, 2006, pp. 1–19.

20. A. N. Burkitt. "A review of the integrate-and-fire neuron model: II. Inhomogeneous synaptic input and network properties". *Biological cybernetics* 95:2, 2006, pp. 97–112.

21. V. Chavez-Demoulin, A. C. Davison, and A. J. McNeil. "Estimating value-at-risk: a point process approach". *Quantitative Finance* 5:2, 2005, pp. 227–234. DOI: 10.1080/14697680500039613. eprint: https://doi.org/10.1080/14697680500039613. URL: https://doi.org/10.1080/14697680500039613.

22. E. Chornoboy, L. Schramm, and A. Karr. "Maximum likelihood identification of neural point process systems". *Biological cybernetics* 59:4, 1988, pp. 265–275.

23. A. C. H. Chow and B. P. Zeigler. "Parallel DEVS: A parallel, hierarchical, modular modeling formalism". In: *Proceedings of Winter Simulation Conference*. IEEE. 1994, pp. 716–722.

24. D. D. Clarke. "Circulation and energy metabolism of the brain". *Basic neurochemistry: Molecular, cellular, and medical aspects*, 1999.

25. C. B. Connor and B. E. Hill. "Three nonhomogeneous Poisson models for the probability of basaltic volcanism: Application to the Yucca Mountain region, Nevada". *Journal of Geophysical Research: Solid Earth* 100:B6, 1995, pp. 10107–10125. DOI: https://doi.org/10.1029/95JB01055. eprint: https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/95JB01055. URL: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/95JB01055.

26. D. R. Cox and P. A. W. Lewis. "Multivariate Point Processes". In: *Volume 3 Contributions to Probability Theory*. University of California Press, 2020, pp. 401–448. DOI: doi:10.1525/9780520375918-024. URL: https://doi.org/10.1525/9780520375918-024.

27. R. Dahlhaus, M. Eichler, and J. Sandkühler. "Identification of synaptic connections in neural ensembles by graphical models". *Journal of Neuroscience Methods* 77:1, 1997, pp. 93–107. ISSN: 0165-0270. DOI: https://doi.org/10.1016/S0165-0270(97)00100-3. URL: https://www.sciencedirect.com/science/article/pii/S0165027097001003.

28. D. J. Daley and D. Vere-Jones. *An introduction to the theory of point processes: volume I: elementary theory and methods*. Springer, 2003, pp. 211–287.

29. D. J. Daley and D. Vere-Jones. *An introduction to the theory of point processes: volume II: general theory and structure*. Springer, 2008, pp. 355–456.

30. A. Dassios and H. Zhao. "Exact simulation of Hawkes process with exponentially decaying intensity". *Electronic Communications in Probability* 18, 2013, pp. 1–13.

31. L. Dematté and D. Prandi. "GPU computing for systems biology". *Briefings in bioinformatics* 11:3, 2010, pp. 323–333.

32. S. DeWeerdt. "How to map the brain". *Nature* 571:7766, 2019, S6–S6.

33. V. Didelez. *Graphical models for event history analysis based on local independence*. Logos-Verlag, 2001.

34. V. Didelez. "Graphical models for marked point processes based on local independence". *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 70:1, 2008, pp. 245–264. DOI: https://doi.org/10.1111/j.1467-9868.2007.00634.x. eprint: https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-9868.2007.00634.x. URL: https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-9868.2007.00634.x.

35. J. Eichenauer and J. Lehn. "A non-linear congruential pseudo random number generator". *Statistische Hefte* 27:1, 1986, pp. 315–326.

36. T. R. Fleming and D. P. Harrington. *Counting processes and survival analysis*. Vol. 169. John Wiley & Sons, 2011.

37. E. Gal, M. London, A. Globerson, S. Ramaswamy, M. W. Reimann, E. Muller, H. Markram, and I. Segev. "Rich cell-type-specific network topology in neocortical microcircuitry". *Nature neuroscience* 20:7, 2017, p. 1004.

38. W. Gerstner and W. M. Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

39. W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski. *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press, 2014.

40. D. T. Gillespie. "A general method for numerically simulating the stochastic time evolution of coupled chemical reactions". *Journal of Computational Physics* 22:4, 1976, pp. 403–434. ISSN: 0021-9991. DOI: https://doi.org/10.1016/0021-9991(76)90041-3. URL: https://www.sciencedirect.com/science/article/pii/0021999176900413.

41. D. T. Gillespie. "Exact stochastic simulation of coupled chemical reactions". *The journal of physical chemistry* 81:25, 1977, pp. 2340–2361.

42. F. Goichon, C. Lauradoux, G. Salagnac, and T. Vuillemin. "Entropy transfers in the Linux Random Number Generator". In: 2012.

43. G. L. Grinblat, H. Ahumada, and E. Kofman. "Quantized state simulation of spiking neural networks". *Simulation* 88:3, 2012, pp. 299–313.

44. J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, et al. "Recent advances in convolutional neural networks". *Pattern Recognition* 77, 2018, pp. 354–377.

45. D. Hansel, G. Mato, C. Meunier, and L. Neltner. "On numerical simulations of integrate-and-fire neural networks". *Neural Computation* 10:2, 1998, pp. 467–483.

46. N. R. Hansen. "Hawkes processes and combinatorial transcriptional regulation". PhD thesis. Citeseer, 1910.

47. R. Haslinger, G. Pipa, and E. Brown. "Discrete time rescaling theorem: determining goodness of fit for discrete time statistical models of neural spiking". *Neural computation* 22:10, 2010, pp. 2477–2506.

48. A. G. Hawkes. "Point spectra of some mutually exciting point processes". *Journal of the Royal Statistical Society: Series B (Methodological)* 33:3, 1971, pp. 438–443.

49. A. G. Hawkes. "Spectra of some self-exciting and mutually exciting point processes". *Biometrika* 58:1, 1971, pp. 83–90.

50. S. Herculano-Houzel. "Scaling of brain metabolism with a fixed energy budget per neuron: implications for neuronal activity, plasticity and evolution". *PloS one* 6:3, 2011, e17514.

51. S. Herculano-Houzel and R. Lent. "Isotropic Fractionator: A Simple, Rapid Method for the Quantification of Total Cell and Neuron Numbers in the Brain". *Journal of Neuroscience* 25:10, 2005, pp. 2518–2521. ISSN: 0270-6474. DOI: 10.1523/JNEUROSCI.4526-04.2005. eprint: https://www.jneurosci.org/content/25/10/2518.full.pdf. URL: https://www.jneurosci.org/content/25/10/2518.

52. S. Herculano-Houzel, B. Mota, and R. Lent. "Cellular scaling rules for rodent brains". *Proceedings of the National Academy of Sciences* 103:32, 2006, pp. 12138–12143. ISSN: 0027-8424. DOI: 10.1073/pnas.0604911103. eprint: https://www.pnas.org/content/103/32/12138.full.pdf. URL: https://www.pnas.org/content/103/32/12138.

53. W. D. Hillis and G. L. Steele. "Data Parallel Algorithms". *Commun. ACM* 29:12, 1986, pp. 1170–1183. ISSN: 0001-0782. DOI: 10.1145/7902.7903. URL: https://doi.org/10.1145/7902.7903.

54. N. Hiratani and T. Fukai. "Redundancy in synaptic connections enables neurons to learn optimally". *Proceedings of the National Academy of Sciences* 115:29, 2018, E6871–E6879. ISSN: 0027-8424. DOI: 10.1073/pnas.1803274115. eprint: https://www.pnas.org/content/115/29/E6871.full.pdf. URL: https://www.pnas.org/content/115/29/E6871.

55. A. L. Hodgkin and A. F. Huxley. "A quantitative description of membrane current and its application to conduction and excitation in nerve". *The Journal of physiology* 117:4, 1952, pp. 500–544.

56. T. Hu, Z. J. Towfic, C. Pehlevan, A. Genkin, and D. B. Chklovskii. "A neuron as a signal processing device". In: *2013 Asilomar Conference on Signals, Systems and Computers*. 2013, pp. 362–366. DOI: 10.1109/ACSSC.2013.6810296.

57. X. Hu and B. Zeigler. *A Proposed DEVS Standard: Model and Simulator Interfaces, Simulator Protocol*. 2008.

58. E. M. Izhikevich and G. M. Edelman. "Large-scale model of mammalian thalamocortical systems". *Proceedings of the national academy of sciences* 105:9, 2008, pp. 3593–3598.

59. E. M. Izhikevich, J. A. Gally, and G. M. Edelman. "Spike-timing Dynamics of Neuronal Groups". *Cerebral Cortex* 14:8, 2004, pp. 933–944. ISSN: 1047-3211. DOI: 10.1093/cercor/bhh053. eprint: https://academic.oup.com/cercor/article-pdf/14/8/933/746882/bhh053.pdf. URL: https://doi.org/10.1093/cercor/bhh053.

60. Izhikevitch. *Simulation of Large-Scale Brain Models*. 2021. URL: https://www.izhikevich.org/human_brain_simulation/Blue_Brain.htm#Simulation%20of%20Large-Scale%20Brain%20Models (visited on 07/05/2021).

61. Izhikevitch. *Simulation of Large-Scale Brain Models*. 2021. URL: https://www.izhikevich.org/human_brain_simulation/why.htm (visited on 07/05/2021).

62. A. K. Jain, J. Mao, and K. M. Mohiuddin. "Artificial neural networks: A tutorial". *Computer* 29:3, 1996, pp. 31–44.

63. D. H. Johnson. "Point process models of single-neuron discharges". *Journal of computational neuroscience* 3:4, 1996, pp. 275–299.

64. D. Johnston. *Random Number Generators—Principles and Practices: A Guide for Engineers and Programmers*. Walter de Gruyter GmbH & Co KG, 2018.

65. S. W. Jones. "On the resting potential of isolated frog sympathetic neurons". *Neuron* 3:2, 1989, pp. 153–161.

66. P. Katz, S. Grillner, R. Wilson, A. Borst, R. Greenspan, G. Buzsáki, K. Martin, E. Marder, W. Kristan, R. Friedrich, and D. " Chklovskii. "Vertebrate versus invertebrate neural circuits". *Current Biology* 23:12, 2013, R504–R506. ISSN: 0960-9822. DOI: https://doi.org/10.1016/j.cub.2013.05.039. URL: https://www.sciencedirect.com/science/article/pii/S0960982213006349.

67. S. Keckler, A. Chang, W. Chatterjee, and W. Dally. "Concurrent event handling through multithreading". *IEEE Transactions on Computers* 48:9, 1999, pp. 903–916. DOI: 10.1109/12.795220.

68. D. G. Kendall. "Stochastic Processes and Population Growth". *Journal of the Royal Statistical Society. Series B (Methodological)* 11:2, 1949, pp. 230–282. ISSN: 00359246. URL: http://www.jstor.org/stable/2984078.

69. A. Khinchin. "Mathematical Methods in the Theory of Queueing". In: *Mathematical Methods in the Theory of Queueing*. Trudy Mat. Inst. Steklov. (English translation (1960). London: Griffin), 1955.

70. J. C. Knight and T. Nowotny. "Larger GPU-accelerated brain simulations with procedural connectivity". *Nature Computational Science* 1:2, 2021, pp. 136–142.

71. D. E. Knuth. *The Art of Computer Programming, vol. 2: Seminumerical Algorithms*. 1981.

72. E. Kofman and S. Junco. "Quantized-state systems: a DEVS Approach for continuous system simulation". *Transactions of The Society for Modeling and Simulation International* 18:3, 2001, pp. 123–132.

73. L. Kostal and R. Kobayashi. "Critical size of neural population for reliable information transmission". *Physical Review E* 100:5, 2019, p. 050401.

74. J. Kruger and F. Aiple. "Multimicroelectrode investigation of monkey striate cortex: spike train correlations in the infragranular layers". *Journal of Neurophysiology* 60:2, 1988. PMID: 3171651, pp. 798–828. DOI: 10.1152/jn.1988.60.2.798. eprint: https://doi.org/10.1152/jn.1988.60.2.798. URL: https://doi.org/10.1152/jn.1988.60.2.798.

75. P. L'Ecuyer. "Combined multiple recursive random number generators". *Operations research* 44:5, 1996, pp. 816–822.

76. P. L'Ecuyer. "Uniform Random Number Generators". In: *International Encyclopedia of Statistical Science*. Ed. by M. Lovric. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 1625–1630. ISBN: 978-3-642-04898-2. DOI: 10.1007/978-3-642-04898-2_602. URL: https://doi.org/10.1007/978-3-642-04898-2_602.

77. P. L'Ecuyer and R. Simard. "TestU01: A C Library for Empirical Testing of Random Number Generators". *ACM Trans. Math. Softw.* 33:4, 2007. ISSN: 0098-3500. DOI: 10.1145/1268776.1268777. URL: https://doi.org/10.1145/1268776.1268777.

78. P. L'Ecuyer. "Random Numbers for Simulation." *Commun. ACM* 33, 1990, pp. 85–97. DOI: 10.1145/84537.84555.

79. A. J. Ladd. "A fast random number generator for stochastic simulations". *Computer Physics Communications* 180:11, 2009, pp. 2140–2142. ISSN: 0010-4655. DOI: https://doi.org/10.1016/j.cpc.2009.06.019. URL: https://www.sciencedirect.com/science/article/pii/S0010465509001994.

80. R. C. Lambert, C. Tuleau-Malot, T. Bessaih, V. Rivoirard, Y. Bouret, N. Leresche, and P. Reynaud-Bouret. "Reconstructing the functional connectivity of multiple spike trains using Hawkes models". *Journal of neuroscience methods* 297, 2018, pp. 9–21.

81. P. Lánskỳ and L. Sacerdote. "The Ornstein–Uhlenbeck neuronal model with signal-dependent noise". *Physics Letters A* 285:3-4, 2001, pp. 132–140.

82. L. Lapique. "Recherches quantitatives sur l'excitation electrique des nerfs traitee comme une polarization." *Journal of Physiology and Pathololgy* 9, 1907, pp. 620–635.

83. V. Lawhern, A. Nikonov, W. Wu, and R. Contreras. "Spike Rate and Spike Timing Contributions to Coding Taste Quality Information in Rat Periphery". *Frontiers in Integrative Neuroscience* 5, 2011, p. 18. ISSN: 1662-5145. DOI: 10.3389/fnint.2011.00018. URL: https://www.frontiersin.org/article/10.3389/fnint.2011.00018.

84. P. Lennie. "The cost of cortical computation". *Current biology* 13:6, 2003, pp. 493–497.

85. P. A. W. Lewis and G. S. Shedler. "Simulation of nonhomogeneous poisson processes by thinning". *Naval Research Logistics Quarterly* 26:3, 1979, pp. 403–413. DOI: https://doi.org/10.1002/nav.3800260304. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nav.3800260304. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800260304.

86. E. Löcherbach. "Spiking neurons: interacting Hawkes processes, mean field limits and oscillations". *ESAIM: Proceedings and Surveys* 60, 2017, pp. 90–103.

87. N. K. Logothetis. "What we can do and what we cannot do with fMRI". *Nature* 453:7197, 2008, pp. 869–878.

88. L. Long and G. Fang. "A review of biologically plausible neuron models for spiking neural networks". *AIAA Infotech@ Aerospace 2010*, 2010, p. 3540.

89. W. Maass. "Networks of spiking neurons: the third generation of neural network models". *Neural networks* 10:9, 1997, pp. 1659–1671.

90. P. Magistretti and I. Allaman. "Brain energy metabolism". In: *Neuroscience in the 21st century: from basic to clinical*. Springer New York, 2013, pp. 1591–1620.

91. Z. Mainen and T. Sejnowski. "Reliability of spike timing in neocortical neurons". *Science* 268:5216, 1995, pp. 1503–1506. ISSN: 0036-8075. DOI: 10.1126/science.7770778. eprint: https://science.sciencemag.org/content/268/5216/1503.full.pdf. URL: https://science.sciencemag.org/content/268/5216/1503.

92. L. Mancini and A. M. Paganoni. "Marked point process models for the admissions of heart failure patients". *Statistical Analysis and Data Mining: The ASA Data Science Journal* 12:2, 2019, pp. 125–135.

93. R. Mansuy and R. Sverdlove. "The origins of the word "martingale"". *Electronic Journal for History of Probability and Statistics* 5:1, 2009, pp. 1–10.

94. G. Markowsky. "The sad history of random bits". *Journal of Cyber Security and Mobility*, 2014, pp. 1–26.

95. H. Markram, E. Muller, S. Ramaswamy, M. W. Reimann, M. Abdellah, C. A. Sanchez, A. Ailamaki, L. Alonso-Nanclares, N. Antille, S. Arsever, et al. "Reconstruction and simulation of neocortical microcircuitry". *Cell* 163:2, 2015, pp. 456–492.

96. G. Marsaglia. "Random numbers fall mainly in the planes". *Proceedings of the National Academy of Sciences of the United States of America* 61:1, 1968, p. 25.

97. M. Matsumoto and T. Nishimura. "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator". *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 8:1, 1998, pp. 3–30.

98. H. Mei and J. Eisner. "The neural hawkes process: A neurally self-modulating multivariate point process". *arXiv preprint arXiv:1612.09328*, 2016.

99. P.-A. Meyer. "Démonstration simplifiée d'un théorème de Knight". fr. *Séminaire de probabilités de Strasbourg* 5, 1971, pp. 191–195. URL: http://www.numdam.org/item/SPS_1971__5__191_0/.

100. G. Mezzadri, F. Flandoli, and P. Reynaud-Bouret. "Some Probabilistic Models of Neural Action Potentials and Neural Networks", 2016.

101. E. K. Miller and J. D. Cohen. "An integrative theory of prefrontal cortex function". *Annual review of neuroscience* 24:1, 2001, pp. 167–202.

102. J. Møller and J. G. Rasmussen. "Perfect simulation of Hawkes processes". *Advances in applied probability* 37:3, 2005, pp. 629–646.

103. A. Morrison, S. Straube, H. E. Plesser, and M. Diesmann. "Exact subthreshold integration with continuous spike times in discrete-time neural network simulations". *Neural computation* 19:1, 2007, pp. 47–79.

104. A. Muzy, B. P. Zeigler, and F. Grammont. "Iterative specification as a modeling and simulation formalism for I/O general systems". *IEEE Systems Journal* 12:3, 2017, pp. 2982–2993.

105. J. Nickolls and W. J. Dally. "The GPU computing era". *IEEE micro* 30:2, 2010, pp. 56–69.

106. Y. Ogata. "On Lewis' simulation method for point processes". *IEEE transactions on information theory* 27:1, 1981, pp. 23–31.

107. Y. Ogata. "Space-time point-process models for earthquake occurrences". *Annals of the Institute of Statistical Mathematics* 50:2, 1998, pp. 379–402.

108. J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips. "GPU computing". *Proceedings of the IEEE* 96:5, 2008, pp. 879–899.

109. B. Pakkenberg and H. J. G. Gundersen. "Total number of neurons and glial cells in human brain nuclei estimated by the disector and the fractionator". *Journal of Microscopy* 150:1, 1988, pp. 1–20. DOI: https://doi.org/10.1111/j.1365-2818.1988.tb04582.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1365-2818.1988.tb04582.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1365-2818.1988.tb04582.x.

110. F. Panneton and P. L'ecuyer. "On the xorshift random number generators". *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 15:4, 2005, pp. 346–361.

111. F. Panneton, P. L'ecuyer, and M. Matsumoto. "Improved long-period generators based on linear recurrences modulo 2". *ACM Transactions on Mathematical Software (TOMS)* 32:1, 2006, pp. 1–16.

112. F. Papangelou. "The conditional intensity of general point processes and an application to line processes". *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete* 28:3, 1974, pp. 207–226.

113. J. Parvizi and S. Kastner. "Human intracranial EEG: promises and limitations". *Nature neuroscience* 21:4, 2018, p. 474.

114. A. Peters. "The selfish brain: Competition for energy resources". *American Journal of Human Biology* 23:1, 2011, pp. 29–34. DOI: https://doi.org/10.1002/ajhb.21106.

115. D. Pfeifer. "Coupling methods in connection with Poisson process approximation". *Mathematical Methods of Operations Research - MATH METHODS OPER RES* 29, 1985, pp. 217–223. DOI: 10.1007/BF01920310.

116. M. Pfeiffer and T. Pfeil. "Deep learning with spiking neurons: opportunities and challenges". *Frontiers in neuroscience* 12, 2018, p. 774.

117. T. C. Phi, A. Muzy, and P. Reynaud-Bouret. "Event-scheduling algorithms with Kalikow decomposition for simulating potentially infinite neuronal networks". *SN Computer Science* 1:1, 2020, pp. 1–10.

118. G. M. Pinna and A. Poigné. "On the nature of events". In: *Mathematical Foundations of Computer Science 1992*. Ed. by I. M. Havel and V. Koubek. Springer Berlin Heidelberg, Berlin, Heidelberg, 1992, pp. 430–441. ISBN: 978-3-540-47291-9.

119. H. E. Plesser and M. Diesmann. "Simplicity and efficiency of integrate-and-fire neuron models". *Neural Computation* 21:2, 2009, pp. 353–359.

120. J. G. Rasmussen. "Temporal point processes: the conditional intensity function". *Lecture Notes, Jan*, 2011.

121. A. Reinhart. "A review of self-exciting spatio-temporal point processes and their applications". *Statistical Science* 33:3, 2018, pp. 299–318.

122. P. Reynaud-Bouret, V. Rivoirard, F. Grammont, and C. Tuleau-Malot. "Goodness-of-fit tests and nonparametric adaptive estimation for spike train analysis". *The Journal of Mathematical Neuroscience* 4:1, 2014, pp. 1–41.

123. P. Reynaud-Bouret, V. Rivoirard, and C. Tuleau-Malot. "Inference of functional connectivity in Neurosciences via Hawkes processes". In: *2013 IEEE Global Conference on Signal and Information Processing*. IEEE. 2013, pp. 317–320. DOI: 10.1109/GlobalSIP.2013.6736879.

124. R. A. Rhoades and D. R. Bell. *Medical physiology: Principles for clinical medicine*. Lippincott Williams & Wilkins, 2012.

125. L. M. Ricciardi and L. Sacerdote. "The Ornstein-Uhlenbeck process as a model for neuronal activity". *Biological cybernetics* 35:1, 1979, pp. 1–9.

126. B. Ripley. "On stationarity and superposition of point processes". *The Annals of Probability*, 1976, pp. 999–1005.

127. S. M. Ross. *Introduction to probability models*. Academic press, 2014, pp. 281–343.

128. S. Rotter and M. Diesmann. "Exact digital simulation of time-invariant linear systems with applications to neuronal modeling". *Biological cybernetics* 81:5, 1999, pp. 381–402.

129. K. Ryan, Z. Lu, and I. A. Meinertzhagen. "The CNS connectome of a tadpole larva of Ciona intestinalis (L.) highlights sidedness in the brain of a chordate sibling". *Elife* 5, 2016, e16962.

130. A. M. Schäfer and H. G. Zimmermann. "Recurrent neural networks are universal approximators". In: *International Conference on Artificial Neural Networks*. Springer. 2006, pp. 632–640.

131. M. Schulz, M.-A. Hermanns, M. Knobloch, K. Mohror, N. T. Hjelm, B. Elis, K. Kraljic, and D. Yang. "The MPI Tool Interfaces: Past, Present, and Future—Capabilities and Prospects". In: *Tools for High Performance Computing 2018/2019*. Springer, 2021, pp. 55–83.

132. A. Schüz. "Neuroanatomy". *Scholarpedia* 3:3, 2008. revision #76788, p. 3158. DOI: 10.4249/scholarpedia.3158.

133. T. Schweder. "Composable Markov Processes". *Journal of Applied Probability* 7, 1970. DOI: 10.2307/3211973.

134. A. Seth. "Granger causality". *Scholarpedia* 2:7, 2007. revision #127333, p. 1667. DOI: 10.4249/scholarpedia.1667.

135. M. J. Shelley and L. Tao. "Efficient and accurate time-stepping schemes for integrate-and-fire neuronal networks". *Journal of Computational Neuroscience* 11:2, 2001, pp. 111–119.

136. J. H. Simpson. "Mapping and manipulating neural circuits in the fly brain". *Advances in genetics* 65, 2009, pp. 79–143.

137. M. Snir. "Technical perspective: The future of MPI". *Communications of the ACM* 61:10, 2018, pp. 105–105.

138. D. L. Snyder and M. I. Miller. *Random point processes in time and space*. Springer Science & Business Media, 2012. ISBN: 978-0-387-97577-1. DOI: 10.1007/978-1-4612-3166-0. URL: https://doi.org/10.1007/978-1-4612-3166-0.

139. X. Tian and K. Benkrid. "Mersenne twister random number generation on FPGA, CPU and GPU". In: *2009 NASA/ESA Conference on Adaptive Hardware and Systems*. IEEE. 2009, pp. 460–464.

140. K. Tocher. *Theory of Modelling and Simulation*. 1977.

141. TOP500.org. *K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect | TOP500*. 2021. URL: https://www.top500.org/system/177232/ (visited on 06/02/2021).

142. Y. Van Tendeloo and H. Vangheluwe. "Introduction to parallel DEVS modelling and simulation." In: *SpringSim (Mod4Sim)*. 2018, pp. 10–1.

143. C. L. Vestergaard and M. Génois. "Temporal gillespie algorithm: Fast simulation of contagion processes on time-varying networks". *PLoS computational biology* 11:10, 2015, e1004579.

144. Z. Wan, W. Taha, and P. Hudak. "Event-Driven FRP". In: *Practical Aspects of Declarative Languages*. Ed. by S. Krishnamurthi and C. R. Ramakrishnan. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002, pp. 155–172. ISBN: 978-3-540-45587-5.

145. S. Watanabe. "On discontinuous additive functionals and Lévy measures of a Markov process". In: *Japanese journal of mathematics: transactions and abstracts*. Vol. 34. The Mathematical Society of Japan. 1964, pp. 53–70.

146. M. E. Watts, R. Pocock, and C. Claudianos. "Brain Energy and Oxygen Metabolism: Emerging Role in Normal Function and Disease". *Frontiers in Molecular Neuroscience* 11, 2018, p. 216. ISSN: 1662-5099. DOI: 10.3389/fnmol.2018.00216. URL: https://www.frontiersin.org/article/10.3389/fnmol.2018.00216.

147. J. G. White, E. Southgate, J. N. Thomson, and S. Brenner. "The structure of the nervous system of the nematode Caenorhabditis elegans". *Philos Trans R Soc Lond B Biol Sci* 314:1165, 1986, pp. 1–340.

148. N. Wirth. *Algorithms + Data Structures = Programs*. 1978.

149. H. Yamaura, J. Igarashi, and T. Yamazaki. "Simulation of a Human-Scale Cerebellar Network Model on the K Computer". *Frontiers in Neuroinformatics* 14, 2020, p. 16. ISSN: 1662-5196. DOI: 10.3389/fninf.2020.00016. URL: https://www.frontiersin.org/article/10.3389/fninf.2020.00016.

150. A. M. Zador. "The basic unit of computation". *Nature neuroscience* 3:11, 2000, pp. 1167–1167.

151. B. P. Zeigler. "DEVS theory of quantized systems". *Advanced simulation technology thrust DARPA contract*, 1998.

152. B. P. Zeigler. *Theory of Modelling and Simulation*. 1984.

153. B. P. Zeigler. "Using the parallel DEVS protocol for general robust simulation with near optimal performance". *Computing in Science & Engineering* 19:3, 2017, pp. 68–77.

154. B. P. Zeigler, A. Muzy, and E. Kofman. *Theory of modeling and simulation: discrete event & iterative system computational foundations*. Academic press, 2018.

155. R. M. Ziff. "Four-tap shift-register-sequence random-number generators". *Computers in Physics* 12:4, 1998, pp. 385–392. DOI: 10.1063/1.168692. eprint: https://aip.scitation.org/doi/pdf/10.1063/1.168692. URL: https://aip.scitation.org/doi/abs/10.1063/1.168692.

156. R. S. Zucker. "Minis: whence and wherefore?" *Neuron* 45:4, 2005, pp. 482–484.