0.1 Introduction

Neurons are cells that are known fo their data processing power when put in a network. They receive, process and emits alectric waves through their axons and dendrites. The emission of such an electric signal is called "spiking".

0.2 Model and background

A popular model for this phenomenon is called Integrate&Fire, which exists in a deterministic and a stochastic form, and is based on an electric model of the membrane, and more precisely of the membrane's potential. There exist many proposals, modifications and extensions of this model, but we are going to focus on the one proposed by Lewis and Rinzel (2003), Ostojic, Brunel and Hakim (2009), that describes the temporal evolution of the potential of the neuron i in a network of N.

$$\begin{cases} \frac{d}{dt}V_i(t) = -\lambda V_i(t) + \frac{\alpha}{N}\sum_j \sum_i \delta_0(t-\tau_k^j) + \frac{\beta}{N}\sum_{j\neq i} V_j(t) + I_i^{ext}(t) + \sigma\mu_i(t) & \text{if } V_i < V_T \\ V_i(t^+) = V_R & \text{if } \mathrm{V}_i \text{ reaches } \mathrm{V}_T \text{ at time t} \end{cases} \tag{1}$$

0.2.1 What the equation is composed of

We are going to describe the components of this equation by explaining the behaviour of a neuron. First the potential, mentionned a bit earlier, that is denoted V in the equation. It varies between two values, V_R and V_T, respectively a reset value and a threshold value. Whenever the potential increases at or over the V_T threshold, it is immediately reset to V_R. This event is unconditionnaly followed by the emission of a spike, which can be viewed as a transmission of the potential from a neuron to others. It is worth noting that the model presented here is incomplete as it does not formally forbid the emission of another spike as it is the case for biological neurons. Indeed, just after spiking a neuron enters in a refractory state, during a predefined period, during which it is impossible for the neuron to spike again. That being said, a refractory period spontaneously emerges from the model *de facto* as the potential of a neuron is reset immediately after spiking. Under this condition, a tremendous amount of external potentials would be needed for a neuron to spike again.
Speaking of external potentials, they are the results of other neuron's spiking in the network, and are modeled here by the Dirac terms in equation (1). Basically, whenever a neuron j in the network of N neurons spikes, the potential of the neuron i in the network receives a *kick* of size $\frac{\alpha}{N}$. Four remarks here: the spikes are instantaneous, the α is constant for the whole system in the equation, the network is a complete graph and the potential of a neuron at a given time

depends on all the previous spikes it has received since the dawn of times (or at least since its creation).
The instantaneous hallmark of the spiking part of the Integrate&Fire model is perfectly assumed, is especially appropriate for extracellularly recorded neural signals. Spikes are stereotyped events: they are commonly viewed as a realization of a 'point process' due to the all-or-none nature of the action potential and its effects on the postsynaptic neurons. On the other hand it is more and more believed that the delay between a spike emission and its receiption plays a role in the encoding of the information. Some models have been conceived to fill this gap, but are not discussed here. If interrested, you can look at [INSERT HERE PUBLICATIONS ABOUT I&F MODELS WITH DELAYS].
About the weight of the *kicks*, they are constnt in the base equation described above, but one of the main goals of this project was to study the case where the weight depends on the neuron. The α then becomes $\alpha^{i,j}$. As a starting point the value of the $\alpha^{i,j}$ is a Bernouilli of a parameter p. In this case the connectivity of the graph decreases, as well as the mean action potential received from presynaptic neurons. This can be compensated by the introduction of a constant term in front of the Bernoulli to increase $\alpha^{i,j}$ value. Actually the parameter p of the Bernoulli is equivalent to a probability of connection between two neurons. This is more realistic as the graph of neurons in the brain is really sparse on average [INSERT VALUES OF CONNEXION IN BRAIN].
An important characteristic of neurons in real brains is the type of signal they transmit: excitatory (the signal increases the action potential of postsynaptic neurons) or inhibitory (the signal decreases the action potential of postsynaptic neurons). In the brain the proportion is about 20% of excitatory neurons, for 80% of inhibitory. The excitatory neurons signals are stronger, compensating the inhibitory signals. This is not modelled in the model, where all neurons are considered excitatory.
The Dirac form of the interaction also allows to not store the full history (or at least a partial one as there is a reset of the potential over a certain value anyway) of spikes. This is important as it allows not to preoccupy about what happened to the network before the beginning of the simulation, and we can then assume that whatever initial state the network is given is its full state. This is not the case for other kinds of point processes, such as Hawkes processes, for which a stationarity hypothesis and a warming up time are necessary in order to have believable data from the simulation [INSERT REFERENCE HERE].

Continuing with the description of the equation terms, there is a dependence of the potential of the neuron i with the potential of all other neurons in the system and with an external electrical current yet to define. Both of them come from the analogy between a neuron's membrane and an electric circuit and from which the Integrate&Fire model has been derived. The external current represents the effects of external parts of the studied system. The effect of a neuron potential on all other neurons potentials comes from the link that exist between neurons. As the graph is considered in this model to be fully connected, all other neurons in the system have an influence on all other neurons. This joint between

neurons (named axons and dendrites) are modeled in the Integrate&Fire view as simple ohmic conductances.
These two terms will be considered as equal to zero during our analysis, for two reasons. First, in terms of modelling, it is fair to consider we are looking at the whole system, and so there is no need for an external input current. This has no sense biologically speaking, as all sensations are thr fruit of electric currents emitted by the neuronal sensors in our body, but all neurons are considered to be their own generators here, and we are not interested in the response to the system to external stimulis but rather in the natural behaviour of the system. [TYPICAL VALUES OF BETA THAT WOULD JUSTIFY CONSIDERING It AS NULL?]. The second reason comes from the reasons of the study of this model. Having immediate spiking and only excitatory interactions in the model introduces a strong possibility of a "blowing up" of the system in finite time. The conditions necessary for the occurrence of this phenomenon has been examined in [INSERT CITATIONS] and in this document. The issue lying in the "jumps" of the system, the external current and the potential dependency turn out to be an unnecessary difficulty in the model, so we will simply not consider them from now on.

Last but not least the two terms that describe the evolution of a neuron's potential, independently from what happens in the rest of the system. The λ term comes from the natural tendency of the membrane to let go some of its accumulated potential. On the other end the $\sigma\mu_i$ term is a noise (actually a white gaussian noise) that is supposed to be independent from neuron to neuron.

0.2.2 The real equation

That being posed, we can now pose the real equation under study in this report.

$$V_t^i = V_0^i + \int_0^t b(V_s^i)ds + \sigma W_t^i + \sum_{j=1}^{N} J^{j\to i} M_t^j - M_t^i(V_T^i - V_R^i)$$

Here b is Lipschitz continuous, and $\forall V, b(V) = -\lambda(V-a), (\lambda, a) \in \mathbb{R}_+$. This function will make the potential drift towards the value a. The J term models the interactions between neurons, their values depends on the number of neurons in the system and which neurons are actually involved in the interaction. Throughout the event several values of interaction have been tested. They generally are Bernoullis variables, fixed as at the creation of the system.
In this paper we are going to focus on what sets of parameters favors the apparition of a blowing up of the system. The influence of the interaction term is a big part of the study, but the influence of other parameters, such as the b function, the noise, and the topology of the network are also looked at.

0.3 The algorithm

Before showing any result a discussion is needed on the algorithmic and the implementation part. The goal is to simulate as many neurons as possible, in order to approach as close as possible the mean field hypothesis. The first discussion is on the threshold. There are indeed two ways of modelling it, as a "hard" threshold or as a "soft" one. Hard here means deterministic, as opposed to soft that means stochastic. the deterministic approach is quite natural, as it was said that neurons' potential shall be reset *immediately* after having crossed the threshold.

This viewpoint is annoying: neurons don't all have the exact same threshold, thus introducing a bit of randomness in the spiking event can count as if we were accounting for the difference. A more major issue, which can be considered as a limitation of the model, is the occurrence of a cascade event that could come directly from the hard threshold. [EXPLICATION DE POURQUOI LE HARD THRESHOLD PROVOQUE CA] The other choice is to consider the neurons as point processes (for instance and in this work as Poisson processes), that have a certain probability of spike as soon as their potential gets over the threshold. The cascade event cannot occur anymore, as no two neurons can spike at the exact same time, while it is believed the blow up phenomenon can still happen, though a new definition may be needed.

0.3.1 The thinner the better

[PLACER UNE BLAGUE SUR LE TITRE] There are two common ways of simulating a stochastic PDE. The Euler-Maruyama method for numerical approximation of stochastic differential equations results in the construction of a Markov chain on the interval $[0, T]$ of simulation. While quite simple to set up and understand, it is wastefull for processes that spend a lot of time not changing, in this case not spiking. In addition, the resolution of the time interval is dependent on the derivative of the variable of interest. Here, as we are interrested in studying extreme variations in the system, the resolution should have been very low in order to not miss relevant information, making the computation time very high for a high number of processes at stake.
The second method is the thinning, which is a common method for draws from distribution of not well defined repartition function [A PRECISER]. Here the stochastic process is the network of neurons. Actually what is going to be sampled here is not the potential itself but a function of the potential. This function will take into account the threshold, as well as influence the neurons in their spiking will. Indeed, the chosen function dilates the difference between the potential of a neuron and its threshold so that any increase in the potential above the threshold reduces a lot the mean time for the next spike and makes the neuron a much better candidate for being the chosen one.
So here is the function of interest:

$$f(V^i(t)) = K * Pos^{\gamma}(V_i(t) - V_T^i) \tag{2}$$

$V_i(t)$ being the potential of the neuron i at time t and V_T^i being the threshold associated with the neuron i. K and γ are constant accross neurons, but an extention of the model may consider making them dependent on the neuron. Typically, $K \approx 10^5$ and $\gamma \approx 5$. The K value inflatuates values when they are positive. This is important as the difference, more still at the power of gamma, between the a potential and the threshold must be as small as possible. The typical value of a potential at a spiking time is well below 1. This is an issue for numerical simulations, as the precision for floating point numbers is clearly not infinite on computers. The same issue arises at several places in the algorithm, where the comparison between the probability function and something else is necessary.

The choice of a good maximum for the sampling procedure is as always critical. The computation cost for actualising the potential is quite high given the equation at stake (exponentials and normal values are not cheap resources). This shall drive the direction chosen for the thinning procedure. There are indeed two choices for the thinning procedure: after having drawn a random time value (length of the jump), either decide which is the spiking neuron first and then whether the jump is accepted, or first decide if the jump is accepted and then decide which neuron is spiking. The second procedure is very interesting for the process of deciding which is the spiking neuron is skipped every time a jump is rejected, making the second procedure more economic *a priori*. A more thorough thought on what information is needed in order to make the computation of whether the jump is accepted or not shows us the exact computation of the potential of every neuron at the potential time of spike is needed, which as mentionned above means a lot of expense of computational time. This explains the choice of the first procedure as for the thinning, which only requires the computation of one potential per jump.

At this stage we have a first draft for the thinning procedure algorithm, but we can go even further. Here the maxima of the probability functions are chosen time-agnostically, but there is a way to refine the difference between the actual potential and its theoretical maximum, that is to look for spikes in given time intervals. This way there are much less possible variations in potential for any neuron, and so the maximum can be much closer of the real value in the whole interval, meaning less rejection during the simulation. Also the variance of the brownian motion is less important, so there are less risks for it to actually take extreme values.

0.3.2 Method for big networks

The algorithm presented above works perfectly well until we consider the memory needed for running it. Even though the full matrix of interactions is not

stored (only the relevant parts), the memory for storing the interaction graph takes around $p * N^2$, where p is the probability of interaction and N the number of neurons. For even 10^5 neurons that makes about 1 GB, given 1 byte is enough to store the index of neurons (and the addresses in memory)(spoiler: it is not) and with a probability of connction of 0.1... where the probability of connection was 1 in the mathematical papers. Of course for a probability of connection of 1 the interaction matrix is not needed, but it is as soon as one want to explore cases where at least some neurons are not connected each other.

Several solutions have been imagined and implemented, we are going to explain them. The basic and most natural one have been seen just above, and we can easily create simplify the algorithm for cases where the number of connections is exactly 1 or 0. In the case the graph would take too much room in the memory, the solution found is to reconstruct the needed parts on the fly. That is to say a deterministic procedure is needed, able to give the user the postsynaptic neurons of a given neuron of the system. Such a procedure is actually quite easy to conceive, and is derived from the algorithm that builds the graph in the first case. The only difference is that the randomly chosen postsynaptic neurons are not stored in a matrix (or whatever is the data structure of choice), the state of the rng is instead (the state before the loop that draws the random postsynaptic neurons). This way it is quite easy to reconstruct parts of the graph: given a neuron the postsynaptic connections of which are wanted, one take the corresponding state of the rng and loops on all the connections (their number is also stored in an array).

0.3.3 Improving speed

But we can go further! While the whole algorithm is not really a good candidate for parallelization, as the state of the system at any time strongly depends on the state of the system the time just before (here the time, and states of the system, is not really continuous as this is a discrete event simulation, so the last event is more accurate). But! There are part of the programm that are perfect candidates for a parallelization. For instance the computation of the maxima, or, in the case we have the interaction graph available, the effect of a spike on the postsynaptic neurons. At this point a lot of the readers will have thought of using openmp [INSERER ALTERNATIVES ET EXPLIQUER POURQUOI OPENMP, BASIQUEMENT C'EST FACILE A UTILISER ET EFFICACE], and this is exactly what has been done. The gain is interesting [15%, MAIS FAIRE UNE ETUDE PLUS POUSSEE].

Hum... Okay, that looks promising, but there are still parts of the programm unoptimised, and unfortunately the heaviest part of all (the one responsible for the reconstruction of parts of the interaction graph) is one of those. As this solution has already been made up to compensate the weakness of the algorithm, this means that whenever it must be used ot would be like hitting a wall in

terms of performances. Thanksfully, if openmp is helpless in that case, the C language is not, and we can call for the standard parrallelization solution implemented in the language. [A COMPLETER LORSQUE CE SERA FAIT.... MARGOULIN !!]

0.4 A definition of the blow up

In the deterministic case, the blow up is quite easily defined by the limit to a time value t of the variations in the spike rate equals to infinity. This

0.5 Test cases

Several test cases where considered, starting from the considerations on the interactions. With the notations posed above:

- $\alpha^{i,j} = \alpha$
- $\alpha^{i,j} = \alpha^i \alpha^j$, which is particularly interesting in the case where α^i or α^j is equal to zero. However, in that case, the interaction matrix is really sparse, as the graph contains a complete part and a set of independent neurons.
- $\alpha^{i,j} = \alpha^i$
- $\alpha^{i,j} = \alpha^j$
- $\alpha^{i,j}$ $\mathbb{B}(p)$
- $\alpha^{i,j}$ $\mathbb{B}(\frac{\ln^{\frac{1}{2},1,2}(N)}{N})$
- Variations in β, the constant of moduling the interactions, especially for the bernouillis

0.6 A word about the rng

Mersenne twister, and not xorshift because the mersenne allows for a reproduction of the simulations, which is also necessary for a part of the algorithm. It is well known, and has a huge period which is important for a part of the algorithm. Also there are a lot of good implementations, some of the creator itself who actively update them when bugs or improvements are found. The method also allow for good generation of random real numbers (not only integers)

A related subject is the distriibution followed by the random numbers. The algorithm presented earlier needs several random numbers from various distributions, none of them being too exotic (uniform numbers with arbitrary upperbound and normal numbers centered on zero with arbitrary variance), but as

the library used only provide uniform random series of 32 or 64 bits and random floating point numbers in [1,2) or [0,1], [0,1), (0,1], (0,1), we must implement a way to generate the numbers drawn from our arbitrary distributions. In both cases a sampling method (again) is used, and we are going to describe them.

First the uniform case. We have a random integer generator, and we assume it gives perfectly random numbers in $0, 1, ..., M$, while we want uniform numbers in $0, 1, ..., U$. A common method to achieve this is to use a modulo: $\mathbb{U}([0, M])mod[U + 1]$. The issue here is that unless M is a multiple of $U + 1$, the uniformity of the numbers is lost using this method. The case in which the method works can be used as a trick for guarrantying uniformity though. Indeed, if M is not a multiple of $U+1$, there must exist one that is lower than M but close anyway. Knowing that, we can simply take any number that is lower than $U + 1$ and in the rare cases a number greater than $U + 1$ is drawn, we just have to reject it and draw another one. Of course the maximum value M must be quite high, compared to $U + 1$ in order for this method to be effective. For instance, if $U + 1 =$, there is a 49% of rejection. If the rest of the euclidean division of M bu $U + 1$ is U, the

0.7 To be added

A word on this maximum value and the brownian motion. It is assumed that the noise will never exceed the assigned maximum value at any moment of the simulation. Yet we do not compute the value of the Brownian motion at any given time, first because it would be quite ineficient and secondly because of the impossibility of such an act! Thus we can only act as if the noise never exceeded the assigned maximum values. The coefficient helps at that, still, in virtue of [Inserer le theoreme pour un chemin C d'un brownien entre deux points, la probabilite du chemin symetrique C est la meme que le chemin C]. So we can say the simulation is at least 50% good.

A jump is usually rejected with a probability [Insert $\frac{max_pot}{\sum max_pot}$], but having the time of the jump higher than the upper bound of the current time interval for computing the maximum value is also a kind of reject. In the first case the simulation updates the potential and time of last spike for the chosen spiking neuron, but does not interfere with its post-synaptic fellas. The simulation then draws a new random number from an exponential distribution with the same parameter as the previous one. In the second case the parameter must be recomputed, thus the maximum values for all neurons must be recomputed. That means the second case costs much more, computationaly speaking, than the first one. Thus a great care must be given to the choice of the size of the time interval.

Nonetheless this choice can only be driven by experimentation (though there are ways to overtake this limitation, see [Insert reference to the pre-computation of a smaller network to speed up the beginning of another network]). Some other choices in the method where driven by reason. First the Marsaglia's method

for generating random normaly distributed numbers. The choice here over a classical Box-Muller method is motivated by two points: the computation cost of trigonometric functions can be high - although that is less true on modern architectures - but more importantly there is a risk of loosing the stability of the computation for very close to zero values of the uniform numbers generated (the probability of which is quite high as we speak about millions if not billions of pseudo-randomly generated numbers).

An important matter is the subject of the floating numbers precision. Indeed if the cumulative sum of the potentials is too high, the exponentials values for the time of next event are very small, potentially smaller than the possible precision.

0.8 Reflexions

TO TALK ABOUT:

- If λ is small enough then the model with drift is very close to the model without, as the drift becomes very small. On the other hand the noise has a very large amplitude, allowing for a neuron to sometimes cross the threshold, especially when other neurons have spiked before (the noise is a gaussian centered on zero so the the potential oscillates around the sum of all the action potentials received since its last spike).
- The tests cases (which, why).
- A test for checking if Poisson or not Poisson?
- A discussion on the rng used (tests with xorshift?)
- Rather than having a probability of spike of 0 below the threshold and $\frac{f(V)}{F_max(V)}$ above, would not it be more interesting to consider a gaussian probability of spike depending on the potential and centered on the threshold?
- Definition of the blow up