ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

Scuola di Scienze
Corso di Laurea Magistrale in Fisica

Fluctuation properties in Random Walks on Networks and Simple Integrate and Fire Models

Relatore:
Prof. Armando Bazzani

Presentata da:
Elena Tea Russo

Sessione II
Anno Accademico 2014/2015

Sinossi

In questa tesi si è studiato l'insorgere di *eventi critici* in un semplice modello neurale del tipo *Integrate and Fire*, basato su processi dinamici stocastici markoviani definiti su una rete. Il segnale neurale elettrico è stato modellato da un flusso di particelle. Si è concentrata l'attenzione sulla fase transiente del sistema, cercando di identificare fenomeni simili alla sincronizzazione neurale, la quale può essere considerata un evento critico. Sono state studiate reti particolarmente semplici, trovando che il modello proposto ha la capacità di produrre effetti "a cascata" nell'attività neurale, dovuti a Self Organized Criticality (auto organizzazione del sistema in stati instabili); questi effetti non vengono invece osservati in Random Walks sulle stesse reti. Si è visto che un piccolo stimolo random è capace di generare nell'attività della rete delle fluttuazioni notevoli, in particolar modo se il sistema si trova in una fase al limite dell'equilibrio. I picchi di attività così rilevati sono stati interpretati come valanghe di segnale neurale, fenomeno riconducibile alla sincronizzazione.

Abstract

In this thesis *critical events* have been studied for a simple *Integrate and Fire* Neuron model based on Markov Stochastic Dynamical Systems on a Network. The electrical neuronal signal is here modelled as a flow of particles. The attention has been focussed on the transient phases of the system, searching for phenomena similar to neuron synchronization, which can be considered a critical event. Very simple Networks have been studied, finding that the proposed model has the capability to produce cascade effects in the neural activity, due to self organized criticality, that are not observed in a Random Walk Dynamics on the same Network. A small random stimulus has been found capable to generate notably large fluctuations in the activity of the network, especially if the systems is at the edge of the equilibrium. These peaks of activity are interpreted as avalanches of neuronal signals.

A mia madre, mio padre
e mio fratello.

Introduction

The study of Complex Systems has a wide range of applications, from physics to biology, including social sciences. [1] [2] The behaviour of a Complex System is influenced by two factors: its basic components and the network of relationships that regulates their interaction. It is a matter of fact that a very small perturbation of a Complex System may change the macroscopic state of the system itself. An important and fascinating field of study that benefits from Complex Systems is Neuroscience, as the neural system is considered a paradigmatic example of Complex Systems. A neural system can be modelled as a Neural Network [8]; the study of these objects is not an easy task, because both the neurons and their interactions are extremely complicated. Biologically accurate models of the neuron and of its synaptic connections are usually dropped in favour of simpler models, such as *Integrate and Fire* (IF) networks. [6] [7]

Many works have been carried on this subject, by introducing a large variety of different IF models. [7] A widely studied IF model, for example, is the *Integrate and Fire oscillator* model [11]. It has been shown that with this model it is easy that neurons synchronize, i.e. that close neurons fire consequentially in a small time interval.

In this thesis I develop a simple *Integrate and Fire* Network model (see Sec. 4.2.1), which defines the dynamic of a set of homogeneous neurons whose connections are determined by a *graph*. I study its transient phases, searching for similar phenomena as the above mentioned synchronization. Even if the studied graph are very simple, the model is complex enough to present a substantial number of these critical events. In particular a small random stimulus can generate notably large fluctuations in the activity of the network: these peaks of activity are interpreted as *avalanches* of inter-neural signals, especially if the system is on the edge of equilibrium.

To tackle complex system's modelling it is possible to take advantage from *Stochastic Dynamical Systems* theory [3] (see Chapter 1). These models introduce an effective description of the *microscopic dynamics* and of the microscopic interactions. This is obtained by means of Stochastic Processes (*mesoscopic models*), and it reduces the complexity of the system by taking into account the dynamics of relevant mesoscopic or macroscopic observables for the considered phenomenon.

I focussed my attention on a particular subset of Stochastic Dynamical Systems: discrete *Markov Chains* [15]. From these, I derive a form of *Master Equation*, and then I evaluate its time-continuous limit. I recall Fokker-Planck Equation and the connection between Stochastic Models and Dissipative Models, given by the Fluctuation Dissipation Theorem. [5]

In chapter 2 I give a definition of *Random Dynamical Network* (**RDN**). This definition is needed because there are several ways to model a Stochastic Dynamical System on a graph, and therefore it is necessary to specify the function of the graph's elements whit respect to the stochastic dynamics. This model is at the base of the Integrate and Fire Networks further considered in this dissertation, and consists of a Markov Chain on a graph. The N nodes of the graph represents the states of the system, while the weight of the (*directed*) links gives the transitions probabilities. We can consider then a single system as a *single particle* moving on the nodes of the graph. This particle can occupy a certain state at a certain time, according to the Markov Chain whose Transition Matrix is associated to the graph's Weights Matrix. We can then study the properties of M non interacting identical systems as the Random Walks of M *particles* on this graphs.

In Sec. 3.2 it is shown how it is possible to relate the existence of an *internal potential* for a graph to the condition of *detailed balance* for a RDN.

In order to describe systems with a dissipation term and a stimulus (*stimulated* network) it has been introduced an *external reservoir*, i.e. it has been added to the graph one node connected to every other node. Particles are sent to the reservoir with transition probability d (dissipation rate) and re-injected on the network with rate SR (see Sec. 3.4).

In Sec. 4.1 is reported a fast review of *Integrate and Fire* models, and in Sec. 4.2.1 I propose a IF network model based on a RDN. I name these RDNs *tRDN*s (threshold Random Dynamical Networks), for their main feature is the introduction of a *threshold interaction* between the particles. Two possible models have been studied: *all firing* and *threshold firing*. If the number of particles in a node is larger than the threshold θ, in the first case all the node's particles can leave it, while in the second case exactly θ particles can leave it. It is then shown that an *all firing* tRDN has a higher probability to have empty nodes than a *threshold firing tRDN*. Moreover, these models have a *reservoir node* to study the effects on macroscopic observables of an external stimulus.

As to better understand *critical events*, I studied two *SandPile* models: the *Cellular-Automata SandPile model* and the *Abelian SandPile model*. These models shares with tRDNs many features, and can then be used as a reference.

The graphs of the tRDNs analysed in this work are very simple, i.e. squared toroidal 2D lattices with uniform connectivities. Such simple models are close both to IF and SandPile models, and are sufficient to examine how a small perturbation on the network's neurons can affect the collective behaviour.

In Sec. 4.5.1 I give a measure for the Network Activity, related to the threshold, and I define an *activity peak*. I then analyse the properties of these peaks in order to characterize critical events, as large and long fluctuations of the network activity.

By comparing the results obtained from a tRDN and a Random Walk on the same graphs, I identify a control parameter, $\lambda = \frac{SR}{Nd}$ (that is, the average number of particles per node), and the range of this parameter that generates an interesting number of *avalanches*.

The above mentioned results have been obtained from simulations of tRDNs done using an original C++ program, ROnDINE, that I developed during this work. In Appendix A are reported its main features.

Contents

Chapter 1

Dynamical Systems

Dynamical Systems (**DS**s) are models used to describe physical systems evolving in time.[3] [1]

The leading idea is that this physical system described by the DS assumes at time t a state $x(t)$ belonging to a set of possible states $\{x^i\}$, and that the *evolution* of the system, i.e. its passage from a state x^i to another one x^j in a certain time, is defined by a *phase flow*, a group of *evolution operators*.

A Dynamical System consists therefore on a set of possible *states* (a **state space** with an invariant measure, or **phase space** in physics), and of an *phase flow*.

Given an initial state $x(0)$, and a certain time T, the system evolves and assumes a series of states $x(t)_{0\leq t\leq T}$, which is known as its *trajectory* in the phase space. A single trajectory, given an initial state, corresponds to a single *realization* of the dynamical system.

What a physicist would like to do is to be able to predict what state will the physical system assume, at a given time T, given an initial state $x(0)$; the "main job" of physics as a predictive science, then, should be to model physical systems by using proper DSs; in other words, to find the most appropriate evolution operators for the system we want to describe.

The classic and deterministic point of view assumes that, if we *perfectly know the state of the system* and if we know every internal and external interaction, we can model its behaviour by using Newtons' Laws and then perfectly determine the trajectory of the system, which will be unique – being the evolution operator a differential operator on $\mathbb{R}^{2d}$. However, it is impossible to have such a perfect knowledge of any system; Laplace remarked this fact by writing of an "infinitely intelligent mathematician" [4] capable of this, which is nowadays known as *Laplace's Daemon*. This observation tells us that, even if the the hypothesis to have a classical and perfectly deterministic world holds, we cannot predict its evolution perfectly.

Our ignorance about the physical system we want to model using a DS can be listed in two categories:

1. Inability to know perfectly the initial state of the system
2. Inability to know perfectly the processes that influence the system evolution

In the first case, we can refer to two different *state indetermination* that can occur in physics: the **measurement error**, that holds for any system, and the **indetermination principle** of quantum mechanical systems. In the second case, we could be ignorant about the physics events driving the processes, or we could be unable to treat mathematically all the interactions.

Anyway, not being a Laplace's Demon doesn't stops us from modelling physical systems with DSs. For example, we can model the possible evolution of a system whose initial state is affected by a measurement error by *propagating* this error during the system's evolution. Error propagation works properly if the evolution of the system does not make the measurement error on the initial state *explode*, increasing with an exponential law: this is the case of *chaotic* systems.

The introduction of *Statistical Mechanics* (**SM**) and the intrinsic indetermination found in quantum systems obliged physicists to surpass Laplace's point of view, and in general to review the idea of prediction itself.

In a SM picture, we model a system composed of a large number of particles. We know how single particles evolve in time, and we might also know how these particles interact. The limit we have is that we don't know the exact state of the system; however, we know something about it, such as its *probability distribution*, which allows us to compute an average, representative, state, and the respective fluctuations. It is then impossible to perform predictions on the exact state of such a system, but we can compute *average values* of certain *observables* and their *fluctuations*.

This is justified in SM because the average behaviour of the microscopical system drives the macroscopic observables; these latter are what is actually measured in a physical experiment, and so they are what we are we interested in.

Another problem arises when we deal with interacting particles in SM: the effective interaction can often be very difficult. The hypothesis assumed to approximate these interactions is the *Mean Field hypothesis*, which essentially states that a single particle is not subject to the interaction with every other particle in the system, but only to a *mean field* produced by it *nearest neighbour*. This mean field is usually a smooth function, chosen as simple to treat as possible.[14]

The main issue appears when the Mean Field hypothesis does not hold. In this case, we are dealing with a **Complex System**. We can define a *Complex System* as a system whose behaviour crucially depends on its details. These kind of systems are very difficult to be studied analytically, because of the critical dependence from their details and our ignorance about these details themselves. As a solution, we introduce *randomness* in the evolution operator, which is model of our ignorance about the "real" evolution of the system.

At this point, it is impossible to make prediction as the one intended by *Laplace*. We can however compute a *probabilistic* evolution, and then perform probabilistic predictions.

1.1 Markov Chains

Markov chains are the most widely used models for many stochastic processes. Their applications ranges from engineering (*queue theory*) and simulations (*Gibbs Sampling*, *Markov chain Monte Carlo*), to *finance*, *human behaviour*, and often occurs in *chemistry's* and *physics'* models.[15]

Obviously, Markov Chains (or *Markov Processes*, or *Markov Systems*), do not represent all the possible RDSs; a general formulation for RDS, (given in Section 1, in 1.2.2) allows many other kinds of systems to exist, an to be studied. Nevertheless, Markov Processes constitute a wide subset of RDSs, large enough to make their study complicated and extremely relevant. To add to generic Markov Chains stronger constraints, or hypothesis, is a common procedure: this allows a simpler study on simpler systems. An important category of these simpler Markov Systems is represented by those processes that can be expressed in term of a *Master Equation*[26]. Among these processes, it is possible to identify another particular class of random processes, i.e. *Diffusion Processes* [27] [25], usually expressed in terms of the *Fokker-Plank Equation.* [24] Master Equation and Fokker-Planck equation describe very particular and constrained situations; their importance is due to the possibility to be solved analytically.

The vast majority of Markov Processes cannot be expressed in terms of Fokker-Plank Equation nor in terms of Master Equation: in this situations, it is required to develop new instruments to face them. Some of those instruments can be borrowed from the theory of Fokker-Planck and Master Equations, as analogies.

1.1.1 Informal Description

A Markov Process operates on a system living in a given *state-space*, and evolves it throughout time. This general definition holds for any evolving system, or general dynamic system. What characterizes a Markov Process is its *transition matrix* (or *stochastic matrix*), that serves as a "one-step evolution operator".

Given an initial state $s(t)$, the next state $s(t + \Delta t)$ is computed by the transition matrix. The elements of the transition matrix, M_{ij}, defines the *probability* for the system to evolve from the state j to the state i. A state, in a Markov Process, is then not represented by a deterministic entity defining the exact state the system is, but by a *probability vector*, or *distribution*, which gives the *probability* of the system to occupy any state. The elements of the *transition matrix* are named *transition rates.*

The most important element of this kind of evolution, for a stochastic system, is hidden in the meaning of the *transition matrix*: this operator computes the new probability vector from the immediately previous one, and with no other information. Because of this feature, a Markov Process is a *memoryless* process: it evolves with no information about its previous states.

1.1.2 Formal Description

The formal description reported here is the one given by J. R. Norris in his book *Markov Chains* [15].

Let us suppose we have a system whose states are represented by the *countable state-space* I, $i \in I$, and let us suppose we have a *measure* on I, $\lambda = (\lambda_i, i \in I) \mid \lambda_i \geq 0 \ \forall \ i$ and $\sum_{i \in I} \lambda_i = 1$, i.e. a *probability distribution.*

We will work with a *probability space* $(\Omega, \mathcal{F}, \mathbb{P})$, with a *random variable* X with values in I, i.e. a function $X : \Omega \to I$.

If we set $\lambda_i = P(X = i) = P(\{\omega \mid X(\omega) = i\})$, then λ defines the *probability distribution of* X.

Thus X models a *random state* which takes the value i with *probability* λ_i.

The *transition rate* is given by the matrix $\mathbf{M} = (M_{ij} : i, j \in I)$, known as the **Transition Matrix** of the process, which is a *stochastic* matrix. Every row $(\mathbf{M}_{ij} : j \in I)$ is a distribution: $\mathbf{M}$ has to be positive and row-normalized with L^1 norm to 1:

$$\sum_{i=1}^{N} \mathbf{M}_{ij} = 1 \qquad \forall\, j = 1, 2...N$$

the elements of the Transition Matrix gives the **transition rates**, being $\mathbf{M}_{ij}$ the probability for the system to evolve from state j to state i.

Discrete Markov Chain

The discrete-time version of Markov Chains is the simplest model to treat and to describe.
$(X_n)_{n\geq 0}$ is a *Markov chain* with *initial distribution* λ and *transition matrix* $\mathbf{M}$ if:

1. X_0 has distribution λ:
$$P(X_0 = i) = \lambda_i$$

2. for $n \geq 0$, the conditional probability to observe X_{n+1} knowing the past history $X_0 = i_0$, $X_1 = i_1$ ... $X_n = i_n$, has distribution $(\mathbf{M}_{ij} : j \in I)$ and is *independent* from X_0,...,X_{n-1}. This can be written as
$$P(X_{n+1} = i_{n+i} \mid X_0 = i_0,\ X_1 = i_1,\ ...\ X_{n-1} = i_{n-1},\ X_n = i_n) = \mathbf{M}_{i_{n+1}\, i_n}$$

Shortly, we say that $(X_n)_{n\geq 0}$ *is Markov*$(\lambda, \mathbf{M})$.

It can be shown that any discrete-time random process $(X_n)_{0\leq n\leq N}$ is *Markov*$(\lambda, \mathbf{M})$ if and only if $\forall\, i_0, ..., i_n \in I$ we have that $P(X_0 = i_0, X_1 = i_1, ..., X_n = i_n) = \mathbf{M}_{i_N\, i_{N-1}} \cdot \mathbf{M}_{i_{N-1}\, i_{N-2}} \cdots \mathbf{M}_{i_1\, i_2} \cdot \mathbf{M}_{i_0\, i_1} \cdot \lambda_{i_0}$

Markov property is a direct consequence of this proposition and of the above given definition of a Markov Chain: in a Markov Process, future is independent of past given the present. Markov Processes are thus *memoryless*: their evolution depends only on the present state.

Homogeneous Markov Chains

If the transition matrix *does not depend on time*, the Markov Chain is said to be *homogenous*. This property is also known as *time homogeneity*, and is related to the fact that the evolution proceeds with the fixed time-step Δt.

Transition Matrix and Probability Vectors

If we have a *finite* number N of possible *states* $i \in I$, we regard distributions and measures λ as N-vectors $\vec{\lambda}$, and the *transition matrix* $\mathbf{M}$ is an $N \times N$ matrix. By applying n times the stochastic matrix $\mathbf{M}$, what we are doing is to *multiply the matrix by itself n times.*

Vectorial representation for finite-state Markov Chains

If we have a finite number N of states $i \in I$ we can represent the probability distribution of a Markov Process at a given time n as a vector. We will then refer to λ as $\vec{\lambda}$

If we consider the *Markov*$(\vec{\lambda}^0, \mathbf{M})$ (initial distribution is λ^0) process, we write the probability distribution at time n as $\vec{\lambda}^n$.

Then, the evolution of the system reads:

$$\vec{\lambda}^1 = \mathbf{M}\vec{\lambda}^0$$

$$\vec{\lambda}^n = \mathbf{M}^n\vec{\lambda}^0$$

The **stationary distribution** $\vec{\lambda}^{st}$ (if it exists) is defined as by:

$$\vec{\lambda}^{st} = \mathbf{M}\vec{\lambda}^{st} \tag{1.1}$$

In terms of linear algebra, equation (1.1) is an *eigenvalue equation* for the matrix $\mathbf{M}$, with eigenstate $\vec{\lambda}^s t$ and eigenvalue 1. Moreover, all the other eigenvectors have eigenvalues in $(-1, 1) \subset \mathbb{R}$

It is possible to show that a certain class of Markov Processes, i.e. *irreducible homogeneous Markov Chains*, always allow the existence of one, and only one, stationary (or *steady*) distribution. *Irreducibility* of a Markov Chain means that the matrix $\mathbf{M}$ cannot be reduced to a diagonal block matrix. If the Markov Chain is made of K separated block, then there will be K eigenvectors satisfying equation (1.1), and eigenvalue 1 will have multiplicity K.

1.2 Deterministic and Random Dynamical Systems

Dynamics studies *collections of self-mappings* of some space. Those collections usually forms a *measurable (semi)group* $\mathbb{T}$, endowed with its σ-algebra $\mathcal{T}$ (more often the Borel σ-algebra $\mathcal{B}(\mathbb{T})$). This (semi)group is what we use to call *time* [3]

This general ideas holds both for *Deterministic* and *Random* dynamical systems. *Time* is usually represented by $\mathbb{R}$ (*continuous time*) or $\mathbb{Z}$ (*discrete time*), or a one-sided version of these *additive* groups, such as $\mathbb{R}^+$ or $\mathbb{Z}^+$.

We generally study **Measurable Dynamical Systems**, intended as *families* of *measurable self-mappings* of a *measurable space* X, *labeled* with *time* $\mathbb{T}$.

This family can be written as $(\theta(t))_{t\in\mathbb{T}}$ and must satisfy the

Flow (or semi-flow) property: $\theta(s+t) = \theta(s)\circ\theta(t) \quad \forall\, s,t\, \in \mathbb{T}$ ($\circ$ means *composition*).

Moreover, if "time is not passing", the system must not evolve: (**identity map**): $\theta(0) = id_\Omega =$ identity on Ω.

These properties are also known as **one-parameter semi group properties**, being $\theta : \mathbb{T} \mapsto X$.

1.2.1 Deterministic Dynamical Systems (DDs)

Dynamical Systems (DSs) can be described in terms of *flows*: in this case the maps $\theta(t)$ represents the *flow*, which is usually described as $\Phi_{t_0}^{t_1}$. Given an initial state for the system, $x_0 \in X$, the evolution (on a time step t) is given by

$$x_{t_1} = \Phi_{t_0}^{t_1}(x_{t_0}) \quad \text{with } t_0 < t_1 \in \mathbb{T}\ .$$

If this is a *deterministic process*, we have a **DDS**.

We can relax the condition $t_0 < t_1$ given that a mapping has to be invertible: if $t_0 > t_1$ we will have the "reverse dynamics"

$$x_{t_1} = \Phi_{t_0}^{t_1}(x_{t_0}) = (\Phi_{t_1}^{t_0})^{-1}(x_{t_0}) \quad \text{with } t_0 > t_1 \in \mathbb{T}\ .$$

If the flow does not depend on the initial state x_t, it is invariant under time translations; these systems are said to be **time-homogeneous**.

In this situation, we can use a lighter notation by just writing

$$x_t = \Phi^t(x_0) \quad \text{with } t \in \mathbb{T}+$$

where x_0 is the initial state. In this case the reverse dynamics reads

$$x_{-t} = \Phi^{-t}(x_0) = (\Phi^t)^{-1}(x_0)\ .$$

If we have a **discrete** system, with time $\mathbb{T} = \mathbb{Z}+$, we can describe systems with with evolution independent from time in terms of *recurrences*. We can state that a single step in the discrete-time system performs an evolution of Δt. In the most general case, we have

$$x_{t_1} = \Phi_{t_0}^{t_1}(x_{t_0}) \quad \text{with } t_1,\ t_2\ in\mathbb{Z}^+$$

If we have a *time-homogeneous* DS (the evolution is independent from t) and being time discrete, we can express the 1-step evolution of the system from x_{t_0} to x_{t_0+1} as $\Phi^{\Delta t}$, the *elementary* evolution. This is the general mapping that evolves a state in the immediate successive one, and because of the time-homogeneity we can write the general evolution from x_{t_0} to x_{t_1} as

$$x_{t_1} = \Phi_{t_0}^{t_1}(x_{t_0}) = (\Phi^{\Delta t})^{t_1 - t_0}(x_{t_0})$$

To express the general evolution after k evolution steps, we write:

$$x_k = (\Phi^{\Delta t})^k(x_0) = \Phi^{k\Delta t}(x_0) \quad \text{with } k \in \mathbb{Z}^+$$

1.2.2 Random DSs

When we turn from deterministic to random dynamical system, we must introduce a *stochastic component* somewhere in the process. What usually is done is to define the RDS mapping on a *probability space.* Roughly, when we perform the evolution, we wont' apply just one predetermined mapping, but we will have a *collection* of possible mapping to choose among. Each mapping of the collection has a given probability to be selected and applied.

Let us consider a *probability space* Ω and time $\mathbb{T} = \mathbb{R}$. ω is a *random event* in Ω, $\omega \in \Omega$.

As we defined the *phase flow*, thus, we can define the **stochastic phase flow**:

$$\Phi_s^t(x, \omega), \text{ with } t,\ s \in \mathbb{R}^n\ , \qquad \omega \in \Omega\ \text{ (probability space)}$$

With the mapping $\omega \to \Phi_\omega$ *measurable.*

We then require, for any *fixed* random event ω, to satisfy the *flow property*:

$$\Phi_u^t \circ \Phi_s^u = \Phi_s^t \quad s \leq u \leq t$$

along with the *identity map*: $\Phi_t^t(x) = x$.

If $s < t$, we have that $\Phi_s^t = (\Phi_t^s)^{-1}$.

Since in this framework we are working in probability spaces, an important role is played by the **expectation values** of states and observables on the states.

An important property that RDSs satisfy is that the *expectation value* of the stochastic phase flow,

$$\mathbb{E}(\Phi_s^t(x, \omega)) = \bar{\Phi}^{(t-s)}(x)$$

is a *deterministic phase flow*; it must then satisfy the flow property:

$$\bar{\Phi}^t(x) \circ \bar{\Phi}^s(x) = \bar{\Phi}^{(t+s)}(x) \qquad \bar{\Phi}^0(x) = x$$

We can set a condition to define *Stationary Random Processes.* As for Stationary Processes, Stationary Random Processes do not depend on the origin of time, being invariant for time translation. This can be formalized by requiring that

$$\forall\, \omega \in \Omega,\ \forall\, u \in \mathbb{T} \qquad \exists\, \omega_u \in \Omega \quad | \quad \omega_u = T_u\omega \quad \wedge \quad \Phi_{s+u}^{t+u}(x, \omega) = \Phi_s^t(x, \omega_u)$$

where T is a *measurable Map*, not depending on s nor on t, which preserves the probability measure in Ω,

$$\mathcal{P}(A) = \mathcal{P}(T_u(A)) \qquad \text{with } A \subseteq \Omega$$

RDSs can be *continuous* or *discrete* in time as DDSs.

1.3 Markov Processes and Markov Property in RDSs

The beforehand given definition of a **Markov Chain** is an example of Random Dynamical System. I gave a discrete-time description, but the continuous Markov Chain can be easily defined and studied too [15].

From now on will continue to use the discrete-time model.

We can define a subset of RDSs, named **Markov Processes**, which can be described in terms of *Markov Chains.*

Markov Processes are basically RDSs with **no memory**. In simple terms, it means that the evolution of the system from time t to time $t + \Delta t$ only depends on the state of the system at time t, and not from what happened before this moment.

This property has no particular meaning on DDs, because these systems realize one and only one trajectory (given an initial state with infinite precision).

In RDSs, we can have several different possibilities for a trajectory, because the event driving it is random and unknown; what is known is the *probability* to realize a particular event. Given a state x_t, the state at $x_{t+\Delta t}$ is not known until the random event is realized. In this situation, the *history* of the system, i.e. the sequence of the states that brought the system at the actual state x_t, could affect the subsequent evolution by affecting the realizations of the random event.

In *Markovian Processes*, we require this to not happen: the expectation value of $x_{t+\Delta t}$ does not depend on the previous states. We say that the history of the system does not affect the evolution, or that the system has no memory.

This lack of memory is known as **Markov Property** (see Sec 1.1.2).

1.3.1 Markov Chains

In terms of RDSs, we can state that Markov Chains are RDSs satisfying *Markov Property* and **Stationary Process Property**.

It is possible to prove that [15][3] any kind of RDSs with Markov Property and Stationary Process Property *is a Markov Chain.*

Then, we can relate the *Transition Matrix* and *state representation*, introduced for Discrete Markov Chains, to the flow operators.

The general RDS evolves from an initial state x_t to x_s as:

$$x_s = \Phi_t^s(x_t, \omega)$$

Being the process *stationary*, the operator Φ does not depend on s or t, but only on $s - t$:

$$x_s = \Phi^{s-t}(x_t, \omega)$$

If we are working with discrete time, $\mathbb{T} = \mathbb{Z}^+$, we have a discrete time-interval Δt, $s - t = k\Delta t$, and then

$$x_s = \Phi^{k\Delta t}(x_t, \omega)$$

and, because the process is stationary, we can reduce

$$\Phi^{k\Delta t}(x, \omega) = (\Phi^{\Delta t})^k(x, \omega)$$

Given the initial state of the system x_0, we can compute any subsequent state after k steps:

$$x_k = (\Phi^{\Delta t})^k(x, \omega)$$

To understand better this notation we must recall what a stochastic phase flow is. The operator $\Phi_s^t(x, \omega)$ can be seen as a collection of *possible* operator among which, throughout the random variable $\omega(t) \in \Omega$, the *effective* evolution is chosen.

Provided this, we can easily connect the stochastic flow phase operator to the *Transition Matrix*.

We consider not any more the *single state at the time t*, $x(t)$, but **the collection of all the possible states** $\{x^i\}_i$. From this, we define a **state vector** $\vec{s}(t)$, being it of finite or infinite dimension, as it follows:

$$s_i(t) = \begin{cases} 1 \text{ if } x(t) = x^i \\ 0 \text{ otherwise} \end{cases}$$

We then define the **probability vector**, $\vec{p}(t)$, as the vector where the i-th entry gives the *probability for the system to be in the state x^i at time t.* Being a probability vector, it must be normalized (with norm L^1) to 1, and positive-valued. If we know the state of the system, x^j, then we have that $p_j(t) = 1 = s_j(t)$, and $p_i(t) = 0 = s_i(t)$ for $i \neq j$.

Let us now give two important **hypotheses**:

1. Time is discrete ($\mathbb{T} = \mathbb{Z}^+$), and a time-step is equal to Δt
2. In a single time-step the system performs one "movement" from one state to another one, according to the transition matrix $\mathbf{M}$. These movements are determined by the **Exchange Operator $\mathbf{A}(t)$**:.

$$s_i(t + \Delta t) = \sum_{j=1}^{N} \mathbf{A}_{ij}(t) s_j(t) \tag{1.2}$$

$\mathbf{A}(t)$ is a single realization of the evolution process. The matrix $\mathbf{M}$ represents then the *average* of the Exchange Operator: $\langle \mathbf{A}_{ij} \rangle = \mathbf{M}_{ij}$ A single realization is randomly generated from the **Transition Matrix M** (defined in Sec. 1.1.2) by using these rules: for every row i of $\mathbf{A}(t)$, we select a state/column j among all the possible states to be setted to be 1, with probability $\mathbf{M}_{ij}$. In order to transform state vectors in state vectors, its entries must be only 1 or 0, and row-wise normalized to 1:

$$\mathbf{A}_{ij}(t) \in \{0; 1\} \qquad \sum_{i=1}^{N} \mathbf{A}_{ij}(t) = 1$$

With these hypotheses, we have fixed the **time scale** of the system at Δt. Because of this, it is more appropriate for the Transition Matrix to *depend* on this time-scale, i.e. $\mathbf{M}_{\Delta t}$. This means that if we want to consider the evolution of a system at different time scales, the matrix $\mathbf{M}(\Delta t)$ changes; in particular, if we want to perform the limit $\lim_{\Delta t \to 0}$, we must pay attention to preserve the flow property $\mathbf{M}(0) = \mathbb{I}$.

The Transition Matrix transforms *probability vectors*:

$$\vec{p}(t + \Delta t) = \mathbf{M}(\Delta t) \vec{p}(t)$$

And, as we have seen in section 1.1, we have that

$$\vec{p}(t + k\Delta t) = \mathbf{M}(\Delta t)^k \vec{p}(t)$$

If we want to perform more than 1 evolution step, say k, we must generate k exchange operators $\mathbf{A}(t)$, $\mathbf{A}(t+\Delta t)$ ... $\mathbf{A}(t+(k-1)\Delta t)$, and apply them as

$$\vec{s}(t+k\Delta t) = \mathbf{A}(t+(k-1)\Delta t)\mathbf{A}(t+(k-2)\Delta t)...\mathbf{A}(t)\vec{s}(t)$$

A *stationary probability vector*, or **stationary distribution**, is what in section 1.1 has been named *stationary state*. This latter name may be misleading, since it does not refer on the state of the system, but on its probability distribution. The system can change its state, but if it is in a *stationary condition*, the probability distribution does not change as the system evolves.

1.4 Non interacting Many Particles Systems

Assuming that we have more than one realization of the same Markov Process concurrently occurring, with discrete time $\mathbb{T} = \mathbb{Z}^+$ for practice, we may consider a single realization as a single **"particle"**: we want then to study the **"Global System"** composed by M particles.

Since all the particles behave the same way, we can say that they are *identical*. If the particles have no label, we say they are *indistinguishable*.

This is a typical set up of a Statistical Mechanics' system.

It is possible to relate a RDS to a Statistical Ensemble, given that it satisfies some constraints.

If we have M identical particles on N possible states, we can define the **Global State Vector** at a given time as the sum of all the State Vectors of each particle:

$$\vec{\mathbf{s}}(t) = \sum_{m=1}^{M} \vec{s}(m,t) \tag{1.3}$$

Being M and N fixed, we have that

$$\sum_{i=0}^{N} \vec{\mathbf{s}}_i(t) = M$$

where $\vec{s}(m,t)$ is the state of the system/particle m at time t.

We can define the **Particle Distribution** as

$$\vec{\rho}(t) = \frac{\vec{\mathbf{s}}(t)}{M}$$

Each single particle changes its state (in a time-step Δt) according to the same Transition Matrix $\mathbf{M}(\Delta t)$.

In general, the **evolution** of the global state vector reads

$$\vec{\mathbf{s}}(t+\Delta t) = \sum_{m=1}^{M} \vec{s}(m, t+\Delta t) = \sum_{m=1}^{M} \mathbf{A}(m,t)\vec{s}(m,t)$$

where $\mathbf{A}(m,t)$ is the exchange operator randomly generated from $\mathbf{M}(\Delta t)$ for the single particle m at time t.

If the particles are **not interacting**, we can consider the evolution of each particle independently and calculate $\vec{\mathbf{s}}(t+\Delta t)$ as

$$\mathbf{s}_i(t+\Delta t) = \sum_{m=1}^{M} \sum_{j=1}^{N} \mathbf{A}_{ij}(m,t) s_j(m,t)$$

At this point we can define the *Global Exchange Operator* as

$$\bar{\mathbf{A}}_{ij}(t) := \frac{1}{\mathbf{s}_j(t)} \sum_{m=1}^{M} \mathbf{A}_{ij}(m,t) s_j(m,t) \qquad (\text{if } \mathbf{s}_j(t) \neq 0) \tag{1.4}$$

and rewrite the evolution of the global system as

$$\mathbf{s}_i(t+\Delta t) = \sum_{j=1}^{N} \bar{\mathbf{A}}_{ij}(t) \mathbf{s}_j(t) \tag{1.5}$$

From its definition, $\bar{\mathbf{A}}(t)$ must satisfy the following properties:

$$\bar{\mathbf{A}}_{ij}(t) \in [0,1] \subset \mathbb{R} \qquad \sum_{i=1}^{N} \bar{\mathbf{A}}_{ij}(t) = 1 \qquad \langle \bar{\mathbf{A}}_{ij} \rangle = \mathbf{M}_{ij}(\Delta t)$$

where $\langle \bar{\mathbf{A}}_{ij} \rangle$ denotes the average value that can take $\bar{\mathbf{A}}_{ij}$, and it is related to the Transition Matrix $\mathbf{M}(\Delta t)$.

From (1.5), we calculate the variation $\mathbf{s}_i(t) - \mathbf{s}_i(t+\Delta t)$ as:

$$\begin{aligned} \mathbf{s}_i(t+\Delta t) - \mathbf{s}_i(t) &= \sum_{j=1}^{N} \bar{\mathbf{A}}_{ij}(t)\mathbf{s}_j(t) - \mathbf{s}_i(t) \\ &= \sum_{j=1}^{N} \bar{\mathbf{A}}_{ij}(t)\mathbf{s}_j(t) - \sum_{j=1}^{N} \bar{\mathbf{A}}_{ji}(t)\mathbf{s}_i(t) \end{aligned} \tag{1.6}$$

$$\Rightarrow \quad \mathbf{s}_i(t+\Delta t) - \mathbf{s}_i(t) = \sum_{j=1}^{N} \Big(\underbrace{\bar{\mathbf{A}}_{ij}(t)\mathbf{s}_j(t)}_{\text{incoming flux}} - \underbrace{\bar{\mathbf{A}}_{ji}(t)\mathbf{s}_i(t)}_{\text{outgoing flux}} \Big)$$

We can then identify an *incoming* and an *outgoing* flux of particles on the state i.
Using the vector notation, we have

$$\begin{aligned} \vec{\mathbf{s}}(t+\Delta t) \; - \; \vec{\mathbf{s}}(t) = \; & \bar{\mathbf{A}}(t)\, \vec{\mathbf{s}}(t) \; - \; \vec{\mathbf{s}}(t) = \\ & (\, \bar{\mathbf{A}}(t) \; - \; \mathbb{I}_N \,)\, \vec{\mathbf{s}}(t) =: \; \Delta\vec{\mathbf{s}}(t+\Delta t) \end{aligned}$$

At this point, by using the properties of $\bar{\mathbf{A}}(t)$, we can write the **average dynamics** from (1.6) as it follows:

$$\langle \mathbf{s}_i(t+\Delta t) \rangle - \langle \mathbf{s}_i(t) \rangle = \sum_{j=1}^{N} \mathbf{M}_{ij}(\Delta t) \langle \mathbf{s}_j(t_0) \rangle - \sum_{j=1}^{N} \mathbf{M}_{ji}(\Delta t) \langle \mathbf{s}_i(t) \rangle$$

And using the vector notation:

$$\begin{aligned} \langle \vec{\mathbf{s}}(t+\Delta t) \rangle \; - \; \langle \vec{\mathbf{s}}(t) \rangle \; = \; & \mathbf{M}(\Delta t)\, \langle \vec{\mathbf{s}}(t) \rangle \; - \; \langle \vec{\mathbf{s}}(t) \rangle = \\ & (\, \mathbf{M}(\Delta t) \; - \; \mathbb{I}_N \,)\, \langle \vec{\mathbf{s}}(t) \rangle =: \; \langle \Delta \vec{\mathbf{s}}\, \rangle (t+\Delta t) \end{aligned} \tag{1.7}$$

1.4.1 Stationary Condition

As said before, the stationary distribution for a *single* Markov memoryless system (in this case, *particle*) m, $\vec{p}_{stat}(m)$, which is given by

$$\vec{p}_{stat}(m) \mid \quad \mathbf{M}(\Delta t) \vec{p}_{stat}(m) = \vec{p}_{stat}(m)$$

The probability to find particle in the i-th state is given by the i-th component of $\vec{p}_{stat}$.

Therefore $\vec{p}_{stat}$ defines an **invariant probability measure** for the single particle stochastic dynamics. If $p_{i,stat}$ is the same for every state i, every state is equivalent. If $p_{i,stat}$ vary with respect to the state i considered, it means that some states have more probabilities to be occupied by the single particle than the others.

If all the M particles are **non-interacting**, we can search for a **global** stationary distribution, by requiring to satisfy a **stationary condition** – i.e. that the probability to find any particle in a certain state is determined only by the state itself. In other words, in a stationary condition the M particles are **indistinguishable**, and the global state does not depend from the initial condition of the system.

Thus, a global system with N single-particle states and M indistinguishable non interactng particles, the probability of the global system to realize a given global state $\tilde{\vec{\mathbf{s}}}$ reads:

$$\mathbf{p}(\tilde{\vec{\mathbf{s}}}, t) = M! \prod_{i=1}^{N} \frac{p_{i,stat}^{\tilde{\mathbf{s}}_i}}{\tilde{\mathbf{s}}_i!} \tag{1.8}$$

which is a *multinomial distribution.* The average number of particles in state i is given by $\langle \mathbf{s}_i \rangle = p_{i,stat}$ and it has variance $var(\mathbf{s}_i) = Mp_{i,stat}(1 - p_{i,stat})$.

1.4.2 Local balance

Let $\vec{e}_i$ be the vector that represent a single particle being in the state i (since particles are indistinguishable, I drop the previous notation specifying the m-th particle). We can write an **elementary particle exchange** from j to i on the state $\vec{\mathbf{s}}$ as $\vec{\mathbf{s}} + \vec{e}_i - \vec{e}_j$; these two states are said to be *connected*, since they follow each other.

In a *stationary condition* the particle distribution satisfies a **local continuity equation** (or **local balance**), since it does not change between two connected states. The variation of the *average current* of particles between two nodes i and j is 0. Therefore we have that

$$\sum_{k=1}^{N} \mathbf{s}_i \mathbf{M}_{ki}(\Delta t)\mathbf{p}(\vec{\mathbf{s}}) = \sum_{j=1}^{N} (\mathbf{s}_j + 1)\mathbf{M}_{ij}(\Delta t)\mathbf{p}(\vec{\mathbf{s}} + \vec{e}_i - \vec{e}_j) \tag{1.9}$$

which can be verified by putting in $\mathbf{p}(\vec{\mathbf{s}})$ and $\mathbf{p}(\vec{\mathbf{s}} + \vec{e}_i - \vec{e}_j)$ according to (1.8).

1.4.3 Detailed Balance

Some particular systems, in a stationary condition, satisfy a stronger equation for the average current of particles, named **detailed balance**. In this case not only the variation of the average current between two nodes is 0, but the average current itself is null.

1.4.4 Evolution Probability

We can compute the probability for the global system to be in state $\vec{\mathbf{s}}(t)$ at time $t + \Delta t$ as the probability it has to evolve to it from any other state $\vec{\mathbf{s}} - \Delta\vec{\mathbf{s}}$, by performing a variation of state $\Delta\vec{\mathbf{s}}$. This can be written as

$$\mathbf{p}(\vec{\mathbf{s}}, t + \Delta t) = \sum_{\Delta\vec{\mathbf{s}}} \mathbf{M}_{\Delta\vec{\mathbf{s}}}(\Delta t) \;\; p(\vec{\mathbf{s}} - \Delta\vec{\mathbf{s}}, t) \tag{1.10}$$

1.4.5 Continuous time limit to Master Equation

From a generic discrete Markov system as the one seen above, we want now to pass to a description in terms of *Master Equation* with *continuous time.* We need then to perform a *continuous time limit*, and in order to do that we need to assume the following asymptotic expansion for the Transition Matrix:

$$\mathbf{M}_{ij}(\Delta t) = \delta_{ij} + \hat{\mathbf{M}}_{ij}\Delta t + o(\Delta t) \qquad \text{with} \quad \sum_{j\neq i} \hat{\mathbf{M}}_{ij} + \hat{\mathbf{M}}_{ii} = 0 \tag{1.11}$$

The latter means that the diagonal of the new matrix $\hat{\mathbf{M}}$ has to be negative. This definition allows to compute the evolution on the limit $\lim_{\Delta t\to 0}$ preserving the flow property, that requires $\mathbf{M}(\Delta t = 0) = \mathbb{I}$.

This expansion assumes that only one particle moves in the (very small) time Δt.

We compute the variation of probability from (1.10) as

$$\mathbf{p}(\vec{\mathbf{s}}, t + \Delta t) - \mathbf{p}(\vec{\mathbf{s}}, t) \ = \ \sum_{\Delta\vec{\mathbf{s}}} \hat{\mathbf{M}}_{\Delta\vec{\mathbf{s}}}\, p(\vec{\mathbf{s}} - \Delta\vec{\mathbf{s}}, t) - \sum_{\Delta\vec{\mathbf{s}}} \hat{\mathbf{M}}_{\Delta\vec{\mathbf{s}}}\, p(\vec{\mathbf{s}}, t)$$

From expansion (1.11), this equation means that in the time-step Δt, for $\Delta t \to 0$ there is only *one* exchange of particle occurring, since the other exchanges have probability of order $O(\Delta t^2)$ and vanishes. For very small Δt, only one particle is moving in the network. This allows us to take the limit

$$\lim_{\Delta t\to 0} \frac{\mathbf{p}(\vec{\mathbf{s}}, t + \Delta t) - \mathbf{p}(\vec{\mathbf{s}}, t)}{\Delta t} \ = \ \lim_{\Delta t\to 0} \left(\sum_{\Delta\vec{\mathbf{s}}} \hat{\mathbf{M}}_{\Delta\vec{\mathbf{s}}}\, p(\vec{\mathbf{s}} - \Delta\vec{\mathbf{s}}, t) - \sum_{\Delta\vec{\mathbf{s}}} \hat{\mathbf{M}}_{\Delta\vec{\mathbf{s}}}\, p(\vec{\mathbf{s}}, t) \right)$$

since $\Delta\vec{\mathbf{s}}$ will be of the type of $\vec{e}_j - \vec{e}_i$.

This finally leads to the **Master Equation**

$$\dot{p}(\vec{\mathbf{s}}, t) = \sum_{i,j=1}^{N} \hat{\mathbf{M}}_{ij}\ (\mathbf{s}_j + 1)\, p(\vec{\mathbf{s}} - \vec{e}_j + \vec{e}_i, t) - \sum_{i,j=1}^{N} \hat{\mathbf{M}}_{ji}\ \mathbf{s}_i\ p(\vec{\mathbf{s}}) \tag{1.12}$$

The Master Equation satisfy the following *continuity equation*

$$\sum_{j=1}^{N} \hat{\mathbf{M}}_{ij}(\mathbf{s}_j + 1)\ p_j(\mathbf{s}_j + 1) p_i(\mathbf{s}_i - 1) \ = \ \sum_{j=1}^{N} \hat{\mathbf{M}}_{ji}\ \mathbf{s}_j\ p_j(\mathbf{s}_i) p_i(\mathbf{s}_j)$$

If we consider not the probability distribution, but the *density* distribution, we can rewrite the continuity equation in terms of flow of particles in a particular realization of the system. This can be done if we have a very large number of particles, since

$$\lim_{M\to\infty} \vec{\rho}_i(t) = \vec{p}_i(t)$$

We identify then $\hat{\mathbf{M}}_{ij}\mathbf{s}_j\rho_j(t)$ as the flow of particles going from node j to node i, and, because of the local balance, we have:

$$\dot{\rho}_i(t) = \sum_{j=1}^{N} \hat{\mathbf{M}}_{ij}\mathbf{s}_j\rho_j(t) - \sum_{j=1}^{N} \hat{\mathbf{M}}_{ji}\mathbf{s}_i\rho_i(t)$$

1.4.6 Laplacian Matrix

The Master Equation (1.12) can be written in terms of the **Laplacian Matrix**[16] [18] $\mathcal{L}$:

$$\dot{\vec{p}}(t) = -\mathcal{L}\, \vec{p}(t) \qquad \text{with} \quad \mathcal{L}_{ij} = -\hat{\mathbf{M}}_{jj}\delta_{jj} - \hat{\mathbf{M}}_{ij}$$

The Laplacian Matrix plays in Markov Chains and in the Master Equation the same role of the *Laplacian Operator* ∇^2, which is fundamental in the description of *diffusion processes* on $\mathbb{R}^3$.

This matrix, given the properties of $\hat{\mathbf{M}}$, has all the elements outside of the diagonal negative, while the diagonal is positive.

The Laplacian Matrix is *positive-semidefinite*, i.e. its eigenvalues are all ≥ 0. The smallest eigenvalue, λ_0, corresponds to 0, and the corresponding eigenvector gives the stationary state of the system; in fact

$$\sum_{j=1}^{N} \mathcal{L}_{ij} p_{j,stat} = 0$$

This eigenvector is also the only eigenvector normalized to 1, while the others eigenvectors are normalized to 0.

If there multiplicity of eigenvalue 0 is larger than one, this means that the Markov Chain is not irreducible. We will see that on Networks this corresponds to a graph made of two separate components.

It is possible to show that the second smallest eigenvalue λ_1 it strictly smaller than 1; it can give us an esteem of the relaxation time for the system, from a transient state to the stationary state:

$$||\vec{\rho}(t) - \vec{p}_{stat}|| \propto \lambda_1 t \tag{1.13}$$

which gives for the system a relaxation time scale of

$$\tau \simeq -ln(\lambda_1)$$

1.4.7 Fokker-Planck Equation

Fokker-Planck equation [24] is related to the description of *diffusion processes*, an it particular it has a strong connection to *Stochastic Differential Equations* (**SDE**s) [29]. SDEs describe the evolution of a physical system perturbed by a *white noise*, i.e. describes a particular subset of RDSs.

We can write a generic linear monodimensional SDE as:

$$dx(t) = -\mu(x,t)dx(t)dt + \sigma(t)dW(t)$$

The first term represents a deterministic differential equation, while the second term represents the perturbation due to the a Wiener Process $W(t)$ [27]. Fokker-Planck equation describes the evolution of the *probability distribution* of such a system; it also corresponds to the *Kolmogorov First Equation* or *Backward equation*,

$$\dot{p}(x,t) = -\frac{\partial}{\partial x}\mu(x)p(x,t) + D(x)\frac{\partial^2}{\partial x^2}p(x,t)$$

μ is the *drift coefficient*, and $D(x) = \frac{\sigma(x)}{2}$ is named *diffusion coefficient.* In practice, the second term is a *diffusion term* deriving from the stochastic nature of the system. This connection is due to the Fluctuation Dissipation theorem [5]

In this conditions, the Laplace Operator can be written as

$$\mathcal{L} = \frac{\partial}{\partial x}\mu(x) - \frac{\partial^2}{\partial x^2}D(x)$$

and allows us to write the evolution of the system as

$$p(x,t) = e^{-\mathcal{L}t}p(x,0)$$

The Fokker-Plank equation describes then a "simple" diffusion process; it is possible, under certain conditions, to approximate the Master Equation to such a diffusion process.

1.5 Creation-Annihilation notation

In the context of Many Particles Systems, it is useful to retrieve (adapting it) a very effective notation used in *second quantization* [30], i.e. the **Creation - Annihilation notation**.

Let us consider a single particle, changing its state according to the transition Matrix $\mathbf{M}(\Delta t)$: a single step of its evolution was written in section 1.3 as (see (1.2))

$$s_i(t+\Delta t) = \sum_{j=1}^{N} \mathbf{A}_{ij}(t)s_j(t)$$

with $\mathbf{A}(t)$ randomly generated from $\mathbf{M}(\Delta t)$.

The operator $\mathbf{A}_{ij}(t)$ has been named *Exchange Operator*, because it changes the state of the particle from a state j to another state i.

In practice, we can state that the exchange operator **removes** a particle from a state j and **creates** one in state i. This operation does not always happen, but it has a certain probability to be performed, given by $\mathbf{M}_{ij}(\Delta t)$.

We can then define the operators of *distruction* a_i and *creation* $a_i^\dagger$ such as:

$$a_i\ \vec{s} := \vec{s}(t) - \vec{e}_i \qquad a_i^\dagger\ \vec{s} := \vec{s}(t) + \vec{e}_i$$

Then we can write the "jump" of a single particle from state j to state i (performed from time t to $t+\Delta t$) as

$$\vec{s}(t+\Delta t) = \vec{s}(t) + \vec{e}_i - \vec{e}_j = a_j a_i^\dagger \vec{s}(t)$$

Obviously, this jump has to be possible, and its realization is governed by the exchange operator, which is random.

We can rewrite the local continuity equation (1.9) as

$$\sum_{k=1}^{N} \mathbf{s}_i \mathbf{M}_{ki}(\Delta t)\mathbf{p}(\vec{\mathbf{s}}) = \sum_{j=1}^{N} (\mathbf{s}_j + 1)\mathbf{M}_{ij}(\Delta t)\mathbf{p}(a_j a_i^\dagger \vec{\mathbf{s}})$$

similarly, the Master Equation (1.12) can be written as:

$$\dot{p}(\vec{\mathbf{s}}, t) = \sum_{i,j=1}^{N} \hat{\mathbf{M}}_{ij}\ (\mathbf{s}_j + 1)\ p(a_j a_i^\dagger \vec{\mathbf{s}}, t) - \sum_{i,j=1}^{N} \hat{\mathbf{M}}_{ji}\ \mathbf{s}_i\ p(\vec{\mathbf{s}}) \tag{1.14}$$

This suggests the introduction of two operators similar to a_i and $a_i^\dagger$, with the difference that, instead of operating on the states, they operate on the *probability* to obtain a certain state.

Such operators, usually written as $\mathbb{E}_i^\dagger$, $\mathbb{E}_i$, are known as **Van Kampen** operators [20], and operates in the following way:

$$\mathbb{E}_j \mathbb{E}_i^\dagger p(\vec{\mathbf{s}}) = p(\vec{\mathbf{s}} + \vec{e}_i - \vec{e}_j)$$

and

$$\mathbb{E}_j \mathbb{E}_i^\dagger \mathbf{s}_j p(\vec{\mathbf{s}}) = (\mathbf{s}_j - 1) p(\vec{\mathbf{s}} + \vec{e}_i - \vec{e}_j)$$

We can rewrite the Master Equation (1.14) as

$$\dot{p}(\vec{\mathbf{s}},t) = \sum_{i,j=1}^{N} \hat{\mathbf{M}}_{ij}\mathbb{E}_j\mathbb{E}_i^{\dagger}\ \mathbf{s}_j\ p(\vec{\mathbf{s}},t) - \sum_{i,j=1}^{N} \hat{\mathbf{M}}_{ji}\ \mathbf{s}_i\ p(\vec{\mathbf{s}})$$

1.6 Interacting Many Particles Systems

I recall now the definition of a global system of M indistinguishable particles that can assume N different states.

From (1.3) we defined the global state, and the non interacting global evolution was given by (1.4).

If the particles **are interacting**, the latter does not hold any more. It is necessary to include the interacting term in the computation of the new state.

1.6.1 Threshold Interaction

The interaction I introduce is a "threshold" interaction.

This kind of choice is motivated by the *Integrate and Fire* neural model (see section 4.2) and the *SandPile* model (see section 4.3), which I will examine later.

Basically, the particles are allowed to change their state if and only if there is a sufficient number of particles in the state they are in. To express this mathematically, I use the Heaviside Step Function, which has been used since the beginning of the Neural Network modelling as *activation function*([21]). It is defined as it follow:

$$\Theta(x) = \begin{cases} 1 \text{ if } x \geq 0 \\ 0 \text{ if } x < 0 \end{cases} \tag{1.15}$$

In this case, the threshold is 0. If we want to set a threshold θ, we use $x - \theta$ instead of x as argument for Θ.

From this, we can write the variation of the state in the global system, as it was described in (1.4), in the case of a fixed threshold θ (which is the same for every state) :

$$\mathbf{s}_i(t + \Delta t) - \mathbf{s}_i(t) = \sum_{j=1}^{N} \bar{\mathbf{A}}_{ij}(t)\mathbf{s}_j(t)\Theta(\mathbf{s}_j - \theta) - \sum_{j=1}^{N} \bar{\mathbf{A}}_{ji}(t)\mathbf{s}_i(t)\Theta(\mathbf{s}_i - \theta)$$

We find again an incoming and an outgoing flow, with the difference that now these flows are governed by both the matrix $\bar{\mathbf{A}}(t)$ and the Heaviside function Θ.

Fixed the threshold θ, we can define the **Activation Vector** $\vec{\mathbf{\Theta}}(t)$ as the vector having as i-th component the Heaviside Step Function computed for the state i and the given θ, at time t:

$$\mathbf{\Theta}_i(t) = \Theta(\mathbf{s}_i - \theta)$$

We can then compute the variation of the global state in a matrix form (cfr. (1.7))

$$\vec{\mathbf{s}}(t + \Delta t) - \vec{\mathbf{s}}(t) = \left(\bar{\mathbf{A}}(t) \; - \; \mathbb{I}_N \right) \left(\vec{\mathbf{\Theta}}(t) \, \mathbb{I}_N \right) \vec{\mathbf{s}}(t)$$

There are then two components regulating the dynamics: the *activation vector*, that *depends on the state of the global system* and determines which particles will actually change their state, and the *transition matrix* $\mathbf{M}$, that drives the evolution of the particles allowed to evolved from the activation vector.

Obviously, the particles cannot be considered indistinguishable any more. It is then not straightforward to compute an average evolution as we have done in the non-interacting case. What pops out is that the *activation vector* plays an important role in the evolution of the global system.

If the number of particles in the global system is larger than $N\theta$ (the average number of particles per state is larger than θ) we recollect the average evolution of the non-interacting system.

Threshold-limited Threshold interaction

Another way to define the threshold interaction is the following:

$$\begin{aligned}\mathbf{s}_i(t+\Delta t)-\mathbf{s}_i(t) =& \sum_{j=1}^{N}\bar{\mathbf{A}}_{ij}(t)\theta\Theta(\mathbf{s}_j-\theta)-\sum_{j=1}^{N}\bar{\mathbf{A}}_{ji}(t)\theta\Theta(\mathbf{s}_i-\theta)\\ =&\theta\sum_{j=1}^{N}\bar{\mathbf{A}}_{ij}(t)\Theta(\mathbf{s}_j-\theta)-\sum_{j=1}^{N}\bar{\mathbf{A}}_{ji}(t)\Theta(\mathbf{s}_i-\theta)\end{aligned}$$

This new definition allows the node to conserve a certain number of particles, and avoids an exchange of a large quantity of particles.

We can note that, in this case, if the number of particles in the global system is larger than $N\theta$, i.e. if the average number of particles per state is larger than θ, the evolution reads:

$$\mathbf{s}_i(t+\Delta t)-\mathbf{s}_i(t)=\theta$$

which is different from the previous case.

Chapter 2

RDSs on Graphs: Random Dynamical Networks (RDNs)

In this thesis I focussed on a particular subset of RDSs, which can be described in terms of *Markov Chains*, i.e. **RDSs on Graphs**.

These systems are characterized by a space of states with a particular "topology", i.e. the one imposed by the graph itself. This means that many results obtained for systems evolving in topological spaces such as $\mathbb{R}^d$ cannot be applied; as a first immediate consequence, it it not possible to derive a Fokker-Planck equation for many RDSs on graphs.

It must be pointed out that there are many different ways to set-up a RDS on a graph. For example, the *edges* of a graph could be *part of the state space* or they could just *define relationship among the elements of the state space.*

It is then very important to define clearly which characteristic of the graph affects the evolution of the system, and how.

In this subsection I will recall very briefly some basic definitions and properties of graphs. A simple and effective description of these objects with a Statistical Mechanics approach is given by Albert-Barabasi in their well known article "Statistical mechanics of complex networks" [33].

Done that, I will specify how a RDSs can evolve on a graph. I will name these systems as **Random Dynamical Networks**.

2.0.2 Graphs

A **graph** is a set of generic entities, *nodes* (or *vertices*), connected among each other by *links* (also known as *edges*).

A graph can be *directed* or *undirected*: in the first case, a *link* represents an oriented relationship between nodes, while in the second case it is a is symmetrical relation. If not all the links are equivalent, every link has its specific **weight**, and we have a *weighted* graph. We define w_{ij} as the weight of the link connecting the node j to the node i. Usually, the weight is restricted to positive values: $w_{ij} \geq 0$; if $w_{ij} = 0$ we consider the respective link as non-existing.

The **degree** of a node i, k_i, is defined as the number of links starting from (*outgoing*) or terminating on (*incoming*) it. If the graph is undirected the distinction between incoming and outgoing links is unnecessary; but if the graph is directed, we can consider separately the degree of incoming links ($k_{i,in}$) and that of outgoing links ($k_{i,out}$), the sum of which gives the 'total' degree: $k_{i,in+out}$. In case links possess also a weight, the degree may be in turn weighed so as to give greater value to most "important" links: in this case, we call it **strength** ($s_i = \sum_j w_{ij}$), where is the weight of the link connecting j to i . As we defined s_i, we can similarly define $s_{i,in}$, $s_{i,out}$, etc... in a directed network.

We say a network is **connected** if there always exist a *path* (a sequence of links) connecting a selected node to each other. If a network is not fully connected, then its adjacence matrix can be reduced in blocks with a proper permutation. The two blocks represents two (or more) distinct networks which are unable to "talk" each other: i.e., if I obtain two blocks (we can call them network A and network B) and I select a node from the network A, there is no path that allows me to reach any node in the network B.

We can define the **shortest path** connecting two nodes as the path that connects those two nodes using the minimal number of links. It can be considered as the most efficient way to connect those two nodes; however, the shortest path could be non-existent (if the network is not fully connected), and it could be non-unique.

Adjacency Matrix and Weights Matrix

A graph can be represented in several different ways: the most useful is the **Adjacency Matrix**.

Let us consider a graph G with N nodes. Its adjacency matrix $\mathbf{Ad}(G)$ is an $N \times N$ matrix; $\mathbf{Ad}(G)_{ij}$ represents the existence of the link connecting the node j with the node i:

$$\mathbf{Ad}(G)_{ij} := \begin{cases} 1 & \text{if } j \text{ is connected to } i \\ 0 & \text{otherwise.} \end{cases}$$

If in the graph there are no cyclic links (that is, links pointing to the same node they depart from), then the adjacence matrix has a null diagonal. If the graph is undirected, then the adjacency matrix is symmetrical.

Given the Adjacency Matrix, if the graph is *weighted* we can introduce the **Weights Matrix**

$$\mathbf{W}(G)_{ij} := w_{ij}.$$

We can roughly say that the Adjacency Matrix defines the space we are working in, while the Weights Matrix defines the **dynamics** of the systems moving on the graph; I will now explain how and why.

2.0.3 Random Dynamical Networks

A **Random Dynamical Network** is a RDS composed by M identical and indistinguishable *particles* which evolution is determined by the *Weights Matrix* of a *graph.*

For practice, we consider the *discrete time* case, with time-step Δt.

Given a graph G with N nodes, we identify each *node i* of the graph as a single-particle *state.* As time passes, the particle will randomly choose a new state according to the **Transition Matrix**, which is defined from the Weights Matrix as it follows:

$$\mathbf{M}(G)_{ij} := \frac{\mathbf{W}(G)_{ij}}{\sum\limits_{i=1}^{N} \mathbf{W}(G)_{ij}}$$

i.e. the Transition Matrix of a graph corresponds to its Weights Matrix row-wise normalized with norm L^1 to 1 (property required in order to be a probability preserving Transition Matrix, see 1.1.2).

As for the Adjacency Matrix, if the graph is undirected the Transition Matrix is symmetrical. However, this is not a frequent situation: in general, we will deal with *directed* graph.

It is straightforward that with this definition we are also defining a *class of equivalence* for graphs, since the same Transition Matrix can be obtained from many different Weights Matrices.

We will only consider graphs with **fixed edges** and **fixed weights**, i.e. *the graph structure does not depend on time.*

With this definition, provided that at each time step Δt the single particle makes 1! movement among the nodes of the graph, we state that the single particle evolves throughout a Homogeneous Markov Process as the systems described on section 1.3.1.

A **Random Dynamical Network** (abbreviated in **RDN**) is then defined as the global system of M particles moving on the graph G according to the Transition Matrix generated by its Weights Matrix.

Given this picture, it should be more clear now why the Adjacency Matrix of G determines the topology of the RDSs, since it defines which states are connected; the Weights Matrix, by defining the Transition Rates, determines the dynamics of the system. On the same topology (defined by $\mathbf{Ad}(G)$ we can define infinite different dynamics thought the weights of the links, and every different dynamics will give different properties.

It is important to note that if the graph is **connected**, then the Markov Chain associated is an *irreducible* chain: I will mainly work with this kind of graphs.

Chapter 3

Random Walks on Networks

If we consider a single particle moving on the Network, we can recall the results and the notation used in section 1.3; we just change our dictionary by calling a *state* as a *node* and the *Global State* as the **Network State**; the evolution of the single particle can be understood as a **Random Walk** [22] on the Graph.

The difference between the classical Random Walk and a **Random Walk on a Network** is due to the underlying topology: the first one is usually defined on $\mathbb{R}^N$, a topological space with all the properties of the case, and allows some important operations such as the continuous-space limit; on a graph, this cannot be always done.

A particular class of graphs that can be connected to spaces as $\mathbb{R}^N$ are **lattices**, from which we can obtain $\mathbb{R}^N$ with a proper limit. If we are instead considering a generic graph, it is quite probable to encounter a *small world* graph [33], which breaks the continuity properties required to pass to a continuous representation.

To support the theory of the next subsections, I will report some result obtained from the *Random Dynamical Network Simulator* **ROnDINE** I developed for this thesis. The programs features are described in Appendix A.

3.1 Stationary Distribution

As we said previously, if we have M **non-interacting** particles moving in the network, in a stationary state their probability distribution will be given by the multinomial distribution written in equation (1.8).

Each node i has probability $p_{i,stat}$ to be occupied by a particle, and this will give the mean density of particles on it. I recall that the stationary state vector of the system corresponds to the eigenvector of $\mathbf{M}(\Delta t)$ having eigenvalue 1 (L^1 normalized to 1): this eigenvector can also be named *first* eigenvector, being it the one with the largest eigenvalue [15].

In the case of a graph, the stationary distribution is given by the first eigenvector of its Transition Matrix $\mathbf{M}(G)$.

As said in section 1.2.2, the stationary distribution gives us an invariant probability measure. We can make every node equivalent by rescaling its particle distribution by the corresponding element of the first eigenvector: in this way, each rescaled particle distribution will have mean 1 and variance proportional to M^{-2}.

In figure 3.1 we can see this relation arise from the two simulation cases.

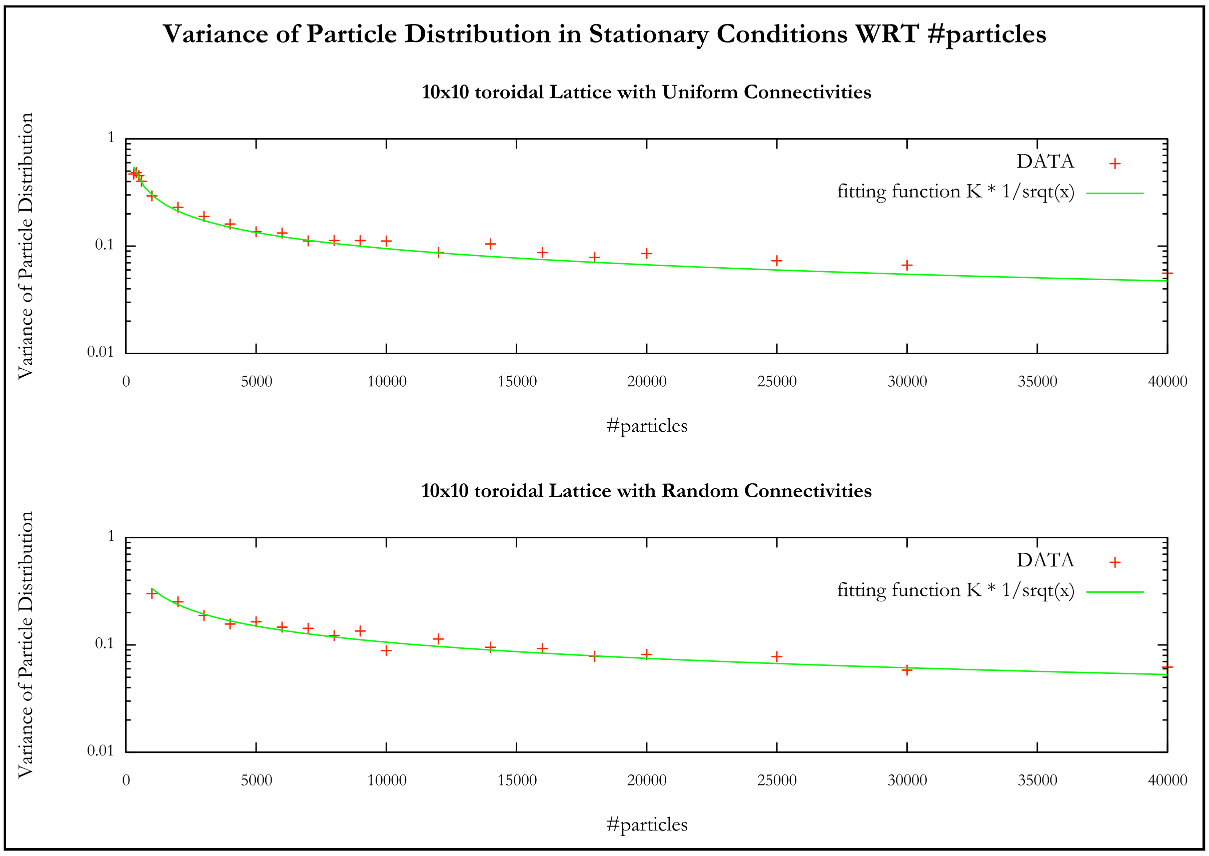


Figure 3.1: Variance of particle distribution in a steady condition w.r.t. M, number of particles in the network. Two cases are shown, both 10x10 notes toridal lattices, in a care with non-wighted links, in another case with randomly weighted links. In any case, the transition matrix does not affect this behaviour.

3.2 Detailed Balance and Internal Potential Energy

An important property for Markov Processes is **detailed balance** (see section 1.4.3).

This property guarantees that there are no circular currents in the network, which could lead to non-physical situations.

To assure detailed balance, we can set on the network an **Internal Potential Energy**, i.e. we assign to each node i a certain potential energy V_i and, given an adjaceny matrix for the graph G, we compute the weight of the link from node i to node j as

$$\mathbf{W}_{ij}(G) \ := \ e^{-\frac{V_i - V_j}{2T}} \tag{3.1}$$

where T plays the role of the temperature of the system, and it is clear that larger is the temperature, more agitated is the system and then more probable are the exchanges of particles between the nodes.

In practice, the transition matrix will be

$$\mathbf{M}_{ij}(G) \propto e^{-\frac{V_i - V_j}{2T}} \tag{3.2}$$

I a stationary condition with detailed balance, we have that

$$\mathbf{M}_{ji}(G) p_{i,stat} = \mathbf{M}_{ij}(G) p_{j,stat}$$

i.e. the currents on the links connecting nodes i and j are the same; it can be rewritten as

$$\frac{\mathbf{M}_{ji}(G)}{\mathbf{M}_{ij}(G)} = \frac{p_{j,stat}}{p_{i,stat}}$$

with the weight defined in (3.2), we can note that the elements of the first eigenvector ($p_{j,stat}$) are indeed given by

$$p_{i,stat} \ \propto \ e^{-\frac{V_i}{2T}}$$

It must be noted that this definition requires at least that the existence of a link from node i to j implies the existence of the link from node j to i; in practice, the adjacency matrix $\mathbf{Ad}(G)$ has to be symmetrical.

The effective value of $p_{i,stat}$ depends also on the connectivity of the node, which determines its accessibility.

3.3 Relaxing Times and multi-timescale systems

As seen in section 1.4.6, we can write a Master Equation in terms of the *laplacian* matrix, and we can see that the system evolves according to

$$p(x,t) = e^{-\mathcal{L}t}p(x,0)$$

The Laplacian Matrix can be defined for *graphs* too, as

$$\mathcal{L}(G) = \mathbf{D}(G) - \mathbf{W}(G)$$

where the matrix $\mathbf{D}(G)$ is defined as

$$\mathbf{D}_{ij}(G) \ := \ \delta_{ij} \sum_{j \neq i} W_{ij}$$

i.e. is the diagonal matrix of the nodes' (weighted) degree.

The evolution of the probability distribution can be intended as a limit for the evolution of the particle distribution of the graph. The Laplacian properties are the same even if it is written for the evolution on a graph, provided that it is an approximation of the effective evolution because of the limited number of particles and the fixed time-evolution scale.

Thus the relaxing times (known also as *mixing times*) are related to the second eigenvalue of $\mathcal{L}$.

3.3.1 Mixing times in Artificial Clustered Networks

Since the time evolution of the system is governed by the second eigenvalue of the Laplacian, it is interesting to analyse the relaxing times of systems having particular properties in its eigenvalues.

In some cases the third, fourth ... eigenvalues of the Laplacian could be extremely close to the second.

This happens when the graph is made of a set of components weakly linked each other. The exchanges of particles inside one of these components has a relaxation time (which, in the paradigm of Markov Processes, is known as **mixing time**) much smaller than that of the whole network, because the exchange of particles among the components is improbable with respect to the internal one.

We can build artificially such kind of networks by setting a proper potential to each node.

Artificial Clustered Network

An ad-hoc built Artificial Clustered Network can be generate as it follows: we imagine to have K_{ext} external components, each of which is composed of K_{int} internal components.

Each external component is composed by K_{int} internal components, which which are Erdos-Reny Random Graphs [23] with K nodes connection probability p_c . Each node of every cluster has potential $V_{int} = 0$.

Then we connect the internal components throughout a *bridge of nodes*, as long as we prefer. For practice, I will consider only one node. These bridges have potential $V_{b,int} > 0$. In this way, the passage of a particle from a cluster to another will be less probable than a movement inside the cluster. I thus generate $K_{int}(K_{int} - 1)$ bridges, and use a single bridge to connect two clusters of the internal component, by linking each node to the given bridge with a probability $p_{b,int}$

A single external component, then, will be composed of $KK_{int} + K_{int}(K_{int} - 1)$ nodes (if we assume that the bridges have only one node). All the components are equivalent.

Now, we connect these external components with the same procedure, generating $K_{ext}(K_{ext}-1)$ external bridges with potential $V_{b,ext} > V_{b,int}$. The external bridges have probability to connect to the internal component's nodes $p_{b,ext}$. The nodes of the external brides does not link to the nodes of the internal brides.

With this construction, we have clearly defined the parameters that drives the dynamics: $V_{b,ext}$, $V_{b,int}$, $p_{b,ext}$,$p_{i,int}$ and p_c; among with the number of components. If we fix the number of components and the connection probability, we can tune the eigenvalues (and then the mixing times).

As an example, I consider the case of 2 clusters of $K = 10$ nodes, with $p_{b,ext} = p_{i,int} = p_c = 1$, two components, and one-node bridges $V_{b,ext} = 10$ and $V_{b,int} = 5$. We will have then 40 internal nodes connected by 3 bridges, 2 internal and 1 external.

The potential setted on the external bridges will characterize λ_1, and the internal bridges will characterize λ_2 and λ_3.

From the computation of such eigenvalues, we obtain:

$$\lambda_1 = 0.999265 \qquad \lambda_2 = \lambda_3 = 0.990227$$

If we compute[1] a single component of the network, i.e. two fully connected clusters of 10 nodes connected with a one-node bridge with potential $V_{b,int} = 5$ we obtain as its second eigenvalue λ_1^5

$$\lambda_1^5 = 0.990962 \simeq \lambda_2 = \lambda_3 = 0.990227$$

the difference is due to the existence of the external bridges in the whole network.

If we instead consider two fully connected clusters of 21 nodes with a one-node bridge with potential $V_{b,ext} = 10$, i.e. a network where the mixing times is due only to the external bridge potential, we obtain as its second eigenvalue λ_1^{10}

$$\lambda_1^{10} = 0.999663 \simeq \lambda_1 = 0.999265$$

which is a bit larger than λ_1 because of the eliminated internal bridge.

We can see the effect of these two mixing times in the artificial clustered network by computing the distance $||\vec{\rho}(t) - \vec{p}_{stat}||$ (see eq. (1.13)) in a simulation of a Random Walk in such a network using ROnDINE.

In order to put the system as far as possible from the stationary condition, I use as initial state $\vec{\rho}(0) = (1, 0, 0, ..., 0)$, i.e. I put all the particles in one node (belonging to the cluster, not on a bridge).

In Fig is shown he distance of the system's particle distribution from the stationary distribution (red points) as time evolves. The green line and the blue line represents two exponential decays with $\tau = -ln(\lambda_1)$ and $\tau = -ln(\lambda_1^5)$. As expected, we first have a relaxation in the inner cluster, characterised by λ_1^5, and for a larger value of t λ_1 becomes dominant. We can justify this behaviour by considering that the initial state is totally asymmetric, and because the external components of the network are weakly connected, we can approximate the relaxation for the first time steps with the relaxation expected by the isolated component, i.e. with $\tau = -ln(\lambda_1^5)$.

We can relate the transition of particles between two clusters to Kramer's Rate Theory [31]. This theory predicts the transition rate of a particle stochastically perturbed from a finite potential well to another one, separated by a potential ΔV. In this case, the transition probability is proportional to $e^{\Delta V}$.

[1]computation have been made using ROnDINE's calculator for eigenvalues and eigenvectors of a graph, which uses LAPACK's DGEEV [19] algorithm contained in *igraph* [12] library

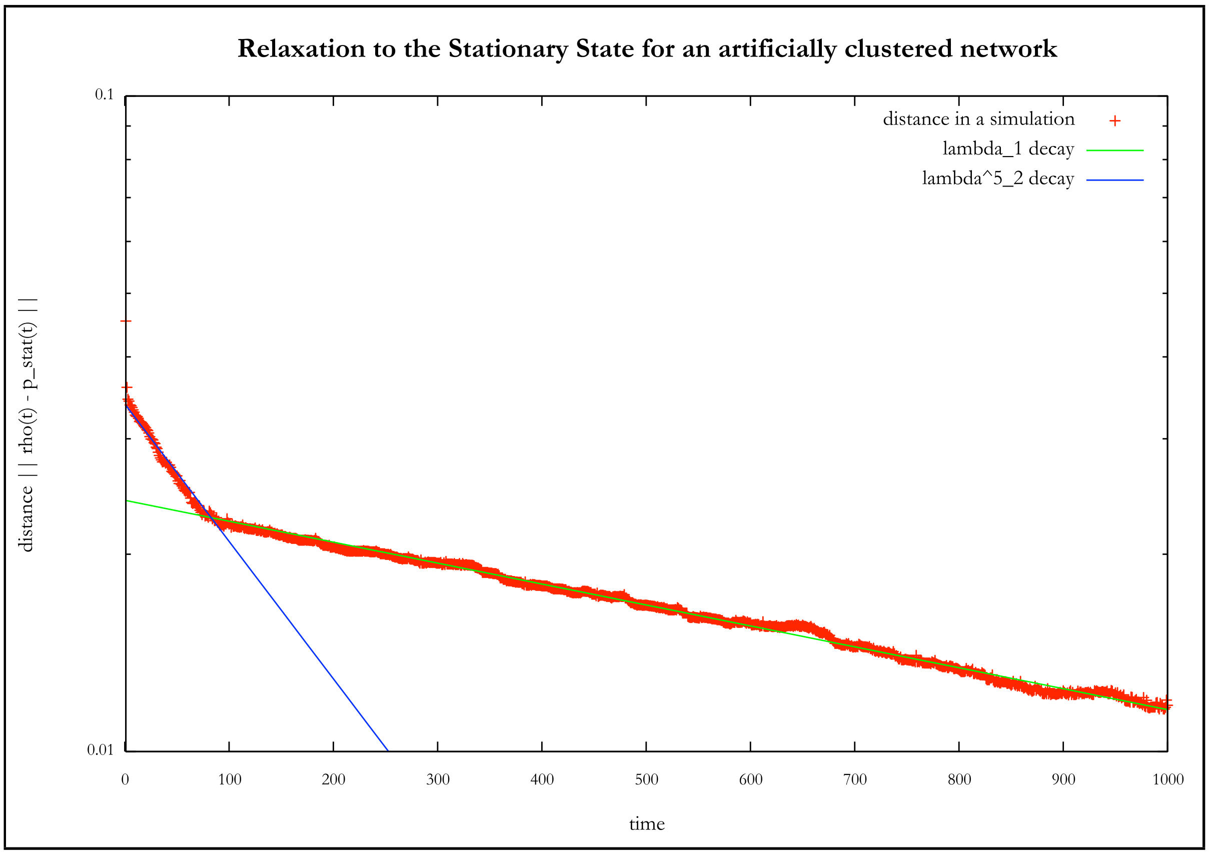


Figure 3.2: $||\vec{\rho}(t) - \vec{p}_{stat}||$ for an artificially clustered network, with 2 clusters of $K = 10$ nodes, $p_{b,ext} = p_{i,int} = p_c = 1$, two components, and one-node bridges with $V_{b,ext} = 10$ and $V_{b,int} = 5$. In order to enhance the transient phase, the initial state is $\vec{\rho}(0) = (1, 0, 0, ..., 0)$. We can clearly see two decay times: the most predominant (fitted with the green line) is the one due to the second eigenvalue of the laplacian matrix, while the initial decay can be related to the second and third eigenvectors. In particular, because of the initial state, we can assume that in the first time steps we have a diffusion due only to the internal bridge, an we can approximate the respective decay time with $\tau = -ln(\lambda_1^5)$.

If we consider an Artificially Clustered Network with $p_{b,ext} = p_{i,int} = p_c = 1$, this relationship is straightforwardly derived from the definition itself of the bridge's weights (see eq. (3.1)): the nodes of the cluster have potential 0, the nodes of the bridges have potential V, then the transition rate from a cluster to a bridge is proportional to e^{-V}.

If we remove the condition $p_{b,ext} = p_{i,int} = p_c = 1$, then the transition rate can change: less nodes of the cluster will be connected to the bridge, and we have to consider also the probability that a particle in the cluster has to reach such a node.

3.4 Perturbation of $\mathbf{M}(G)$: dissipation and stimulus

We want now to connect the network to an external thermostat, which introduces and subtract particles with a constant rate.

We can to this by adding a node in the network, and by connecting it to the rest of the nodes.

We initially consider the $(N+1)\times(N+1)$ matrix $(\tilde{M})(G)$ defined as

$$(\tilde{M})_{ij}(G) \;:=\; \begin{cases} \mathbf{M}_{ij} \text{ if } 1 \;\leq\; i,j \;\leq\; N \\ 0 \text{ for } i,j \;=\; N+1 \end{cases}$$

This matrix describes the initial graph G with the addition of an isolated node. Being composed of two unconnected components, its leading eigenvalue will have multiplicity 1.

We then connect the isolated node, which will be the thermostat, or reservoir. In order to have a dissipation, we set $(\tilde{M})_{ij}(G)$ as it follows:

$$(\tilde{M})_{ij}(G) \;:=\; \begin{cases} \frac{\mathbf{M}_{ij}}{1-d} \text{ if } i,j\; 1 \;\leq\; i,j \;\leq\; N \\ d \text{ for } i,j \;=\; N+1 \end{cases}$$

where d is the *dissipation term*, $0 < d < 1$. With this definition each particle in the network has probability do be dissipated d.

This matrix obviously has as stationary state the vector $\vec{e}_{N+1}$, because every particle will, even if in a long time, reach the reservoir.

We then have two different choices to introduce the *stimulus*. We may want to stimulate one fixed node, or we may want to stimulate a random node for each time step, distributing randomly the stimulus. Morevoer, we may stimulate more nodes concurrently. This will change the dynamics of the system, and I will explore it in section 4.5.

What I want to point out here is that we fix the number of average outgoing particles from the reservoir in a single time-step, to be $\langle P_{res}\rangle = SR$. The fluctuations are due to the quantization of particles.

Being the d the probability to dissipate one particle in a time step and SR the probability to inject a particle in a time step, we can easily recover the average number of particles in the network $\langle P_n\rangle$ when this is in equilibrium with the reservoir from differential equation

$$\frac{d\langle P_n\rangle}{dt} = SR\, dt - d\, dt = \; 0$$

Which is satisfied for

$$\langle P_n\rangle \;=\; \frac{SR}{d}$$

Given this, even if the reservoir and the network are in equilibrium this does not mean that the network is in a stationary condition; the continuous stimulus due to the reservoir could sensibly affect the state of the network.

I will investigate some properties of such stimulated random dynamical networks, in the specific case of a *non linear* dynamics due to the introduction of a *threshold term.*

Chapter 4

Non-Linear RDN and critical phenomena

There are several ways to introduce a non linearity in our system. As said before, I decided to use a threshold interaction (see section 1.6), which is a typical characteristic of a series of models producing *critical events* [2].

I start with a consideration on non interacting systems, such as Random Walks on Networks. In this case, after a long enough evolution, the state of the network reaches a particle distribution that satisfies the multinomial distribution computed from the stationary state of the system. Once in this stationary situation, the fluctuations of the particle distribution are related to $M^{-1/2}$, being M the number of particles; many arguments of Statistical Mechanics apply and it is possible to compute many observables of the system with smaller fluctuations as bigger it is the number of particles, and to take the thermodynamic limit.

These systems are ergodic - which means that the observables computed averaging over long time evolution gives the same value as the observables computed over many contemporary evolutions of a single system. Moreover, they are *strongly* ergodic: the "long time evolution" required to perform a good average computation on the system is not particularly long.

Once we inserted an interaction between the particles on a network, or in general between Stochastic Systems, as we have seen in section 1.4 it becomes difficult to compute the average state and its distribution.

What is more common, when analysing this kind on processes, is to study the distribution of the **fluctuations** of some observable: the result usually shows that *critical phenomena*, i.e. events that in a non-interacting situation were very improbable, becomes much more probable.

If we can roughly say that in a non interacting system the "driving" distribution is a multinomial distribution, which leads to *Poisson* distributions or *Gaussian* distribution, in the interacting case this is not true.

Thus, if in a Random Walk we expect the distributions to have *exponential-like* tails (which is the case of the family of Binomial, Poissonian and Gaussian distribution), the interacting case usually presents distributions with **fat** tails.

The idea to use a *threshold* as interaction is inspired from several models, in particular the **SandPile model** and many **neural dynamics** models.

The SandPile model can be considered a *toy model*, simple in its formulation but with a very interesting variety of results: nevertheless, it is not as simple to study as its simple formulation may suggest. It is very useful and evocative in order to understand what an *avalanche* is, and to model it in the simplest way. I will then use this model as a support for mine, which however wants to be closer to another kind of models, *Neural Models*.

I will begin with a description of the simplest Neural Models proposed in the last century; then I will propose a very simple Integrate and Fire model [6] strictly related to the RDNs previously presented. I will then analyse the SandPile models and relate them to my model..

At last, I will analyse the results obtained from the simulations performed using ROnDINE and compare them with the expected results.

4.1 Neural Models

Neural Models constitute a big class of models, varying from the Rosenblatt's Perceptron [21] to the more biologically accurate Hodgkin and Huxley's Neuron [32].

Some of the most used (and in some case, simplest) models are listed in the **Integrate and Fire** models family [6], originally proposed by Lapique in 1907 [34]. These models basically defines the behaviour of a single neuron as an integrator of electric signal, while doesn't specify their connection. There are several ways to define a Neural Network of IF neurons, and in order to do that is is necessary to specify at least three features:

- If and how the fired signal changes when it travels to another neuron throughout a synapses
- If the response of two a neurons to the same stimulus can be different (i.e., if they have different *threshold potentials*)
- If the signals can be only excitatory or also inhibitory

A typical example of a IF network model are *integrate-and-fire oscillators* [7]. In this particular situation, the activity of a neuron has a certain rhythm, making it similar to an oscillator. [11] These systems are a matter of interest in biological models because the can represent a wide range of other biological systems, such as muscle cells. Many studies have focussed on study these simple models on simple networks, such as lattices, observing the effect of *synchronization*. Two neurons are said to be synchronous if they are firing almost at the same time; this can happen if the transmission time is notably smaller than the recharging time. Neurons in practice synchronise their activity, and generate a macroscopic pulse. This pulse can be intended as an *avalanche of signal*, and such avalanches have been observed in real systems, being them acknowledged as the basis of neural systems functioning.[2] The objective of this thesis is to analyse fluctuations of a very simple IF Network based on RDNs, searching for similar phenomena, being them observed in several other different types of IF networks [9] [10]. In particular, am interested in the properties exhibited from a *static neural network*, i.e. a neural network that is not *learning*. I require this also because in a RDN model a graph cannot graphs.

In the following section I will give a fast review of Integrate and Fire neuron models, in order to understand better the successively proposed model and make comparisons.

4.1.1 Integrate-and-fire models

Integrate and Fire models (**IF**) assumes that neurons communicate throughout an *electrical current*, which travels via synapses and reaches the neuron's core: here, the neuron works as an electrical RC circuit, made up of a parallel capacitor and resistor, which represents the electrical properties of the neuron's cellular membrane.

The neuron is treated as an "integration" device: it receives input currents from his neighbours, and it sums them. When (and if) the sum (integral) of the received *"stimuli"* becomes larger than the neuron's *threshold*, the neuron *"fires"* and transmits a current to its synapses according to its inner potential, gained in the integration process.

Modern models can be explicated in this way: [35]

$$C\frac{dV(t)}{dt} = I_l(V) + I_s(V,t) + I_e(t)$$

Where

- I_s is the *synaptic current* (a noisy-current that is always present in the synapses) ;

- I_e is the *external current* (directly injected in the neuron, i.e. the input current, or **stimulus**);
- I_l is the ***leak** current*;
- C is the *membrane capacitance.*

A *threshold* V_{th} is present in the neuron: when the *membrane potential*$V(t)$ reaches the threshold value, it fires, and it resets its potential to its *rest value* V_r.

The model described above is known to be the general Integrate and Fire model.
The main variants of this model are:

- **SIF**, *simplest integrate and fire* model, has no leaky I_l. If there are no inputs or noisy currents ($I_e = 0$ and $I_s = 0$), the neuron will never change its state.
- **LIF**, *leaky integrate and fire* model, which has a leaky current set to $I_l = g(V - V_L)$, (where V_L is the resting potential $V_L = -70mV$). Such a model is also named "forgetful" model, because the leaky current tends to pull the membrane potential to V_L, vanishing in time the effects of the integration.

Both these variants can be expressed in one general formula for the membrane potential:

$$\tau_m \frac{dV(t)}{dt} = f(V) + I_s(V,t) + I_e(t)$$

Where $tau_m = C$, is the *membrane time constant*, g (if present) is absorbed by the currents, and $f(V) = 0$ for SIF model, $f(V) = -V$ for the LIF model. As said before, when $V(t)$ reachers the threshold V_{th}, the neuron fires and resets its potential to V_r

Some literature defines the **PIF**[36] (*perfect integrate and fire*) model as a SIF model without stochastic current noise: this is the most basic IF model possible.

In this case, the membrane equation reduces to:

$$\tau_m \frac{dV(t)}{dt} = I_e(t)$$

when $V(t) \geq V_{th}$ *fire*, and $V(t) \to V_r$

Described how the core of a single neuron (the membrane) works, we must define two other features: how the neuron relaxes to V_r once he has fired, and how the fired stimulus propagates throughout the synapses.

When we move from the single neuron to a whole network of interconnected neurons, there is another thing that must be taken in account, *synaptic weights.* When a spike travels from a neuron to another one throughout a synapse, the latter modifies the intensity of the stimulus: this modification can occur in several ways.

4.2 RDN modeling IFl

The model I propose is similar to an IF model, but as a very simplified version.

4.2.1 Simple Model Description

This model consists in a *Random Dynamical Network* with *threshold* (**tRND**), connected to an external reservoir (see section 3.4) which both serves as a basin for the dissipated particles and as a stimulator.

Each node in the network has a *fixed* threshold θ. Once the number of particles in a node becomes greater than the threshold, it *fires* the particles out, and distribute them among its neighbours according to the graph's transition matrix. The number of particles fired from the network can be θ (as described in section 1.6) (**threshold firing**) or it could be $\mathbf{s}_i$, i.e. *all* the particles it has (**all firing**). This possibility defines two variations of the model.

We require the stimulus to not be larger than the threshold. In order to avoid this, it is distributed among a sufficient number of nodes, randomly chosen.

For example, if we fix the stimulus to be 11, and the threshold 4, at each time step $11 \div 4 = 3$ nodes are randomly chosen in the network, and the 11 particles are randomly distributed among these nodes.

4.2.2 Comparison with IF models

With respect to IF models, the model I propose has some differences.

First of all, the model I simulate is a *discrete* model: instead of the integration of a continuous current, I have the sum of discrete particles; moreover, firing is instantaneous.

In any case, we can set this analogy:

- A node in the RDN is a **neuron**: more precisely, it is the *neuron cellular membrane* where potential/current accumulates.
- The links of the graph corresponds to the synapses connecting the neuronal cellular membranes. Connections are directed: an edge from node i to node j allows a spike produced from node i to travel to node j.
- The current analysed can be thought in terms of current of particles.
- The membrane potential is intended as the sum of particles in the neuron. The threshold thus defines when the neuron will spike.
- The **threshold** θ is the *same* for every node
- The number of particles that the neuron spikes may vary. Here I distinguish between two sub-models: **all-firing** and **over-threshold** firing.
- Fired particles travel throughout the outgoing links of the neuron, and moves as they do in non-interacting RDN. Thus, we can consider edges' weights as synaptic weights; this automatically excludes the possibility to have *inhibitory* synapses: there can be only *excitatory* stimuli (particles can be sent to other neurons, but not taken. A neuron can loose particles only if it fires them or dissipate the.).
- Since the graphs we use are *static*, we have *no synapitc pasticisty* in this model.
- synapsis are 100% efficent in signal transmission.

- there is no synaptic noise.
- the introduction of the reservoir, and then of the *dissipation* term, means I'm considering a *leaky noise* model.

4.3 The SandPile Model

The SandPile Model[38][39] (**SPM**)is a toy model used to investigate *self-organized* criticality in complex systems.[38][2].

The basic idea of **SPM** is the following: we have a pile of sand, and we add a grain of sand at a random point. The more sand we add, the more we bring the system close to the *edge of stability.* At a certain point, the add of a single grain of sand will make the system activate: the friction will be not able any more to maintain the system in its state, and it will result in a small toppling. If only a small part of the system was unstable, the toppling stops; but if a larger part of the system is at the edge of stability, the few grains moved by that single toppling can generate many new topplings, and in this way create an *avalanche.* Once the system is activated, it continues to topple until it reaches a new stability.

This degree of stability, in real sandpiles, is determined by a *critical slope* for the pile ϑ_c (which depends on the quality of the - dry - sand used). If we start with a sandpile with a slope smaller than ϑ_c, the response of the system will be weak: maybe local topplings may happen, but they are mostly isolated. On the contrary, with a slope *larger* than ϑ_c, big events can happen – with the formation of big avalanches. With such a slope our system is on the edge of stability, which allows the system to show this behaviour.

In 1987 Bak, Tang and Wiesenfeld[40] noted that the construction of a sandpile from zero on a finite table brings the system in a critical state. The fact that this critical state, at the boundary between stability and instability, is built by the system itself is a typical example of *self organized criticality.*

The system being in this critical state shows correlations much larger than what is the lenght scale of the interaction in an equilibrium situation. This means that a small signal can propagate towards all the system, even if an equilibrium condition this can not happen. This is a behaviour similar to what happens during a phase-state transition, and we can identify the system being at the edge of stability as being at the edge of a phase transition.

This means that we can observe these critical phenomena in equilibrium systems only with a certain fine-tuned set of parameters for he system.

4.3.1 Cellular-Automata SandPile model

The simplest SandPile model has been proposed by Bak, Tang and Wiesenfeld[40] and is based on a cellular-automata, an extremely simple DS on a 2D-lattice with a finite number of nodes and discrete-time evolution.

Each node (or site) of the lattice has a state, i.e. the number of particles on it. A toppling-threshold $z_c$is fixed, usually 4. The dynamic reads as it follows: at each time Δt, a particle is added to the system ad a random site. If the site reaches the threshold value, it topples and distribute its particles among its neighbours. If one of the neighbours reached or surpassed the threshold, it topples too. If a toppling-node is on the edge of the lattice (the lattice is not toroidal), then it "dissipate" one particle (two if it is in a corner). In a single Δt the system relaxes, by resolving all the possible toppling: the new state is then a stable state, and the next particle can be added.

Thus, the *relaxation* is much faster than the *particle adding* time.

What we are interested in are *events involving many toppling*, or a*avalanches.* We can measure an avalanche by its length s, i.e. the number of sites it has activated during its "fall". These kind of events happen very often in this kind of systems, and the probability to have avalanches show a power-law like distribution.

4.3.2 Abelian Sand Pile Model

This model is more general than the cellular automata, since it assumes that the dynamics is happening not in a simple lattice but in a more complicated *graph*.

Suppose we have graph G with N nodes, adjacency matrix $\mathbf{Ad}(G)$ and weight matrix $\mathbf{W}(G)$, and we distribute a certain number of particles among all the nodes (this is a setup very similar to a RDN).

The dynamics is driven by a $N \times N$ integer matrix, $\mathbf{L}(G)$, that is very close to a *Laplacian Matrix.* $\mathbf{L}(G)$ has the following properties:

- $\mathbf{L}_{ii}(G) > 0 \; \forall i$
- $\mathbf{L}_{ij}(G) \leq 0 \; \forall i \neq j$
- $\sum_j \mathbf{L}_{ij}(G) \geq 0 \; \forall i$, except for at least one site i such that $\sum_j \mathbf{L}_{ij}(G) > 0$, a *dissipative site.* Moreover, we want that the dissipative site is reachable from every other site; and guarantees that the avalanches do not continue forever.

Thus, at each time step Δt we randomly chose a site, and we increase its number of particle by one (similarly to what the *source* did in section 3.4).

If a node has a number of particles grater than a certain threshold, it topples, and loses a number of particles equal to the number of nodes it is connected to (if it has enough). The basic dynamics reads:

$$\text{if } \mathbf{s}_i > z_{c,i} \;\Rightarrow\; \forall j \; \mathbf{s}_j \longrightarrow \mathbf{s}_j - \mathbf{L}_{ij}(G)$$

(note that the number of particles in j is increasing, since $\mathbf{L}_{ij}(G)$ must be positive defined.)

The threshold is here different from node to node, and it can be setted to be $z_{c,i} = -\mathbf{L}_{ii}(G)$.

The matrix $\mathbf{L}_{ij}(G)$, as it is defined, is very similar to a laplacian matrix, a part from the fact that it has to be an integer matrix and for the existence of (at least) one purely dissipative node that deletes particles – while the evolution given from a laplacian matrix preserves the number of particles in the network.

Avalanches can be characterized by their *size*, i.e. the number of sites that have toppled in the single event. The relative frequency of events size has a distribution cha characterized by a long power law tail, limited by the lattice's size L.

Abelian property

These models satisfy an Abelian property, based on the fact that the order used to add K particles to the net does not affect the final state.

We use in here situation too a *creation-annihilation* paradigm, by defining the operator $\hat{a}_i^\dagger$ as the operator that adds a particle to the site i.

If we have a certain (stable) configuration of the network, $\vec{\mathbf{s}}(t)$, and we add a particle to the system in the node i, we will write the new state as $\vec{\mathbf{s}}(t + \Delta t) = \hat{a}_i^\dagger \vec{\mathbf{s}}(t)$.

It is possible to demonstrate that the operators $\hat{a}_i^\dagger$ and $\hat{a}_j^\dagger$ *commute*, i.e.

$$\hat{a}_j^\dagger \hat{a}_i^\dagger \vec{\mathbf{s}}(t) = \hat{a}_i^\dagger \hat{a}_j^\dagger \vec{\mathbf{s}}(t)$$

In case the addition of a particle to a node generates a toppling, it can be expressed in terms of there operators as it follows:

$$\hat{a}_i^{\mathbf{L}_{ii}} = \prod_{j \neq i} \hat{a}_j^{\mathbf{L}_{ij}}$$

These operators are the generators of a finite abelian semi-group subject to 4.3.2.

With these models the structure of the graphs becomes relevant to determine how large can be an avalanche.

The response of the system to small perturbations can be measured by the probability that, added a single particle ad a certain site, another site at distance r is affected from it. This can be measured by the *two-point correlation function* $G_{ij}(r)$, which can be used to characterize the length of avalanches.

4.3.3 Comparison with tRDN model

The evolution of a SandPile model is similar to the tRDN model. The main difference is the relaxing time scale. In a SandPile model, the systems relaxes to a stable configuration, by dissipating a certain number of particles that can change; while on tRDN model the particles are added in the network continuously, and continuously dissipated, with a *source rate* S and a *dissipation rate* d. Moreover, in a SandPile model usually only one node (or the edge of a lattice) dissipates, while in the tRDN model every node has the same probability to dissipate a particle.

However, these features are required to have a model close to IF model, in particular *leaky* IF.

4.4 Summary of the models

In table 4.1 I report a Summary of the above described models, with their main characteristics.

	Threshold	**Produced Signal**	**Stimulus**	**Dissipation**
RW	No	$\mathbf{s}_i(t)$	No	No
Stimulated RW	No	$\mathbf{s}_i(t)$	rate SR, randomly distributed	d, constant for every node
IFl model	different for every neuron	the charged current is fired (depends on the model)	not specified	constant for every neuron
SandPile model (2D lattice)	4	4 grains	Yes, small and rare	only on the edges
SandPile model (Graph)	outgoing degree of the node	1 grain for neighbour	Yes, small and rare	Yes, in at least one node
***fire-all* tRDN**	$\theta = const$ (4)	$\mathbf{s}_i(t)$	rate SR, randomly distributed	d, constant for every node
***threshold firing* tRDN**	$\theta = const$ (4)	θ	rate SR, randomly distributed	d, constant for every node

Table 4.1: Main characteristics of the models presented so far: Random Walk on Network (see Sec. 3), stimulated Random Walk on Network (see Sec. 3.4), Integrate and Fire model (see Sec. 4.2), Cellular Automata (or 2D lattice) SandPile Model (see Sec (4.3.1), Abelian SandPile Model (or SandPile model on a Graph) (see Sec. 4.3.2), *fire-all* tRDN (see Sec.s 1.6.1 and 4.2.1) and *threshold-firing* tRDN (see Sec.s 1.6.1) and 4.2.1)

4.5 Fluctuations in lattice tRDN

I analyse now the behaviour of a tRDN.

I recall the fact that a tRDN is not only a RDN with a threshold dynamics, but it also has a reservoir node that furnish the stimulus to the network and collects the dissipated particles.

As seen in section 3.4, the ratio between the source rate S and the dissipation d gives the average number of particles in the network $\langle P_n \rangle$.

Fixed a dissipation d, then, we can change $\langle P_n \rangle$ by tuning S. This will also change the average number of particles per node, with respect to the network analysed.

In this thesis I focused in analysing very simple networks, such as **toroidal lattices with uniform connectivities**, as to understand the basic properties of tRDNs and their fluctuation behaviour. In this model, I set the threshold to be $\theta = 4$, being 4 the degree of each node. In this way the proposed model ha the same threshold a lattice SandPile model usually has (see section 4.3).

4.5.1 Activation of a network

I focussed my attention on the **activation of the network**, because of affinities to *neuronal activity* and *sand pile* topplings.

In terms of *neural networks*, a neuron is active when it fires; similarly, a site on a SandPile model is active when it topples, sending particles to its neighbours.

I define then, in a tRDN, an *active node* as a node that is losing particles.

The *activation of the network* at time t $\mathcal{A}(t)$ is defined as the average number of active nodes in the network at time t.

If there is a reservoir, the activity of this node is not considered, since it is external to the network. I remark, moreover, that the threshold dynamics does not apply to particles arriving or departing from the node; which basically means that the dissipation and the stimulus are not influenced by the threshold – as it is in the presented modes above.

It is straightforward that with this definition a Random Walk on a Network has $\mathcal{A}(t) = 1 \ \forall \ t$, provided that there is at least one particle in each node.

If we want to compare the dynamics of Random Walks on Networks with a tRDN, we cannot use this definition, but it is necessary to slightly review it.

We note that in a tRDN the nodes activate if and only if there are at least θ particles in the node. We can then define the activation of a tRDN with threshold θ as the average number of nodes having at least θ particles at the given time:

$$\mathcal{A}_\theta(t) \ := \ \frac{1}{N} \sum_{i=1}^{N} \Theta(\mathbf{s}_i(t) - \theta)$$

(see equation (1.15)). Then, we will compare a tRDN with threshold θ with a Random Walk on the same network by using $\mathcal{A}_\theta(t)$.

4.5.2 Mean Activation analysis

Given a stimulated tRDN with $\langle P_n \rangle$ much larger than θN, we have seen that we basically recollect the dynamics of Random Walk. In the case of a *fire-all* tRDN, this will be the same behaviour of a Random Walk with $\langle P_n \rangle$ particles in the network, while in the *threshold firing* they will be θN. This assertion holds less if the elements of the transition matrix are very inhomogeneous; however, in the simple case of a 2D toroidal lattice, it is straightforward.

We can see from the evaluation of $\mathcal{A}_\theta(t)$ how this phase transition occurs.

In a non interacting situation, the probability to have k particles in a node is given by the Poissonian distribution

$$P(\mathbf{s}_i = k) = \frac{e^\lambda \lambda^k}{k!}$$

where λ is the probability to find a single particle in a node, and being the network I am working with uniform toroidal lattices it is the same for every node; in a stimulated RDN it can be computed as $\lambda = \frac{\langle P_n \rangle}{N}$

Then the average number of active nodes, provided that we have an average number of particles on the network $\langle P_n \rangle$, is computed by

$$P(\mathbf{s}_i \geq \theta) = 1 - \sum_{k=1}^{\theta-1} \frac{e^{\frac{\langle P_n \rangle}{N}} (\frac{\langle P_n \rangle}{N})^k}{k!}$$

With $\theta = 4$, we have

$$P(\mathbf{s}_i \geq 4) = 1 - e^{\frac{\langle P_n \rangle}{N}} \left(\frac{\langle P_n \rangle}{N} + \frac{(\frac{\langle P_n \rangle}{N})^2}{2} + \frac{(\frac{\langle P_n \rangle}{N})^3}{6} \right) \tag{4.1}$$

The mean activation of a a Random Walk on a stimulated Network shoud coincide with this expression; while the one computed for a tRDN should differ, because of the different dynamics.

Case: 10 x 10 uniform toroidal lattice

In Figure 4.1 I report the values obtained for the average activation function $\langle \mathcal{A}_4(t) \rangle$ in a series of simulations performed with *ROnDINE*.

The mean values are computed on 50000 time-steps, after other 50000 time steps of system evolution that guarantees the relaxation of the system to a stationary (or similar to stationary, in the case of the tRDN) condition.

Three curves are shown in Figure 4.1, representing the dependance of $\langle \mathcal{A}_4(t) \rangle$ with the mean value of particles in a node (λ). The first one, in red, is obtained from a threshold-limited tRDN ($\theta = 4$), the second (green) for a Random Walk in a stimulated Network in the same set up, while the third (blue) is the expected value computed using the expression (4.1) .

As expected the second and the third curves are almost superimposed, while the first one differ. In particular, it can be seen that it differs more in the range of $\lambda < 2.5$. This suggest us a window where it is more interesting to investigate.

An interesting and notably different behaviour can be seen in in Figure 4.2, where the same curve is reported for the *all-fire* tRDN. The activity of the network stops at a value that is slightly grater than 0.5.

In this case, the network reaches a particular kind of state, that we can name *chessboard*-like: only half of the nodes are occupied, while the rest are empty; occupied and empty nodes success each other like in a chess board (see Figure 4.3). This chessboard is slightly imperfect due to the injection of particles of the stimulus; being the stimulus big, it stimulates more than one node (up to 3 in the case reported) with 4 particle: the effect is a bit enlargement of the network's mean activity.

This is a very peculiar behaviour and it is due to the network. In particular, this phenomena arises in small lattices with a **even** number of nodes. The mechanism is intuitive: at a certain point, a firing node is surrounded by *charged* nodes, i.e. nodes with 3 particles. The central nodes fires, and it becomes empty; since it fires on average one particle to each neighbour, its neighbourhood, at the successive time

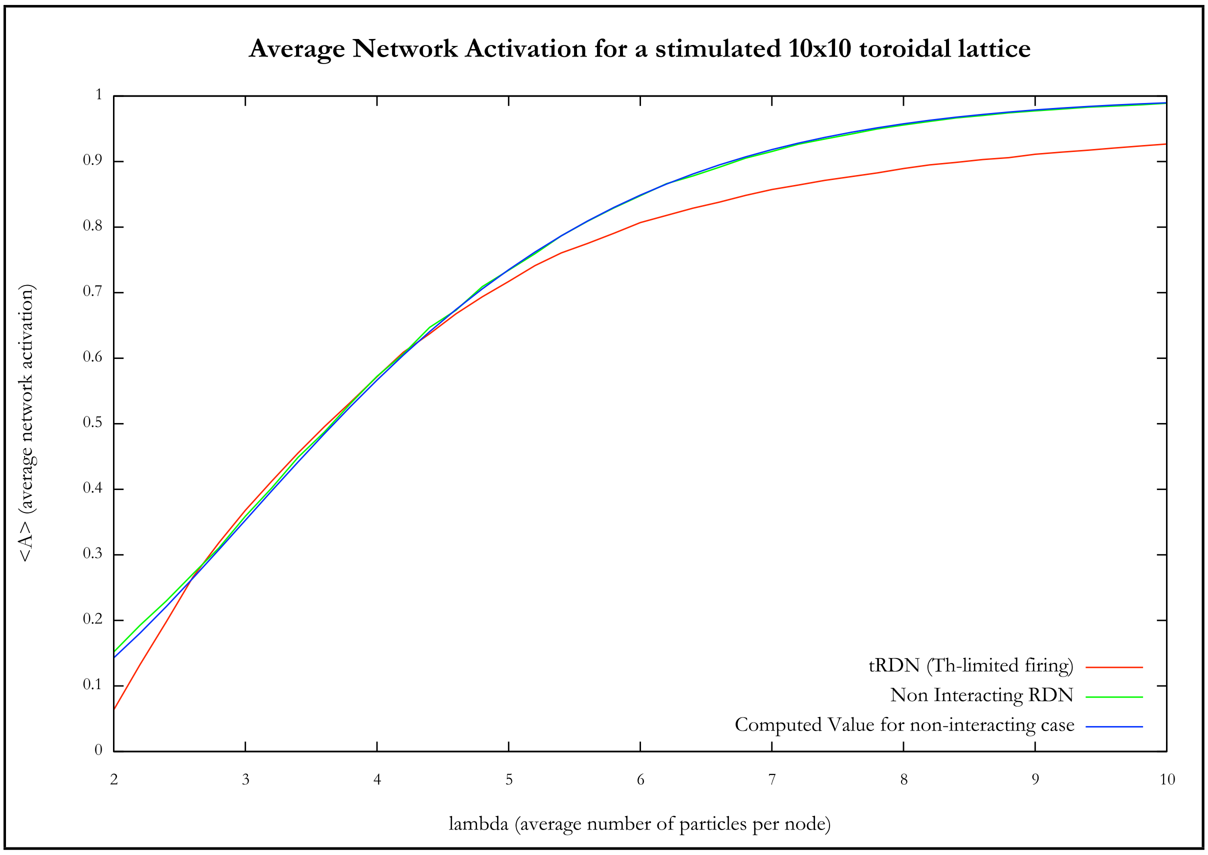


Figure 4.1: Average Network activation $\langle \mathcal{A}_4 \rangle$ for a stimulated 10x10 toroidal lattice. Parameters used are: $d = 0.05$, SR vary. In the x axis is reported the average number of particles per node produced by different SR. Three curves are shown: the first is obtained from a *threshold-limited* tRDN ($\theta = 4$), the second for a Random Walk on a stimulated Network in the same set up, while the third is the expected value computed using the expression (4.1). Average values obtained from 5ffl10^4 time-steps, in a relaxed condition.

step, will fire too. In this way, the 4 neighbours will become empty, and the central node will be charged again. If the network is small enough, this type of dynamics can easily propagate, and, being the number of nodes in the lattice even, it stabilizes. If the number of nodes is even, at the borders this effect does not propagate, because the nodes are in a counter-phase. Such a dynamic can activate all the network with a chessboard-like particle distribution just if $\langle P_n \rangle > \theta N/2$; if the dissipation rate is small and the source rate big enough (but not too big, or it risks to activate too many nodes at the same time), the network "prefers" this behaviour, because of its higher lever of order to any other one possible in such conditions.

In the *threshold-limited* case, this doesn't happen because the limitation on the fired particles does not allow the node to become empty as frequently as in the *fire-all* case.

As it can be see, f the mean activity for the stimulated tRDNs tends to 0 as $\langle P_n \rangle \rightarrow 0$.

If $\langle P_n \rangle << \theta$, the tRDN becomes almost inactive, because the small probability to have nodes sufficiently charged. We can just see some rare spike in $\mathcal{A}_\theta(t)$, due to some locally firing node, which is unable to propagate its signal being it surrounded by almost empty nodes.

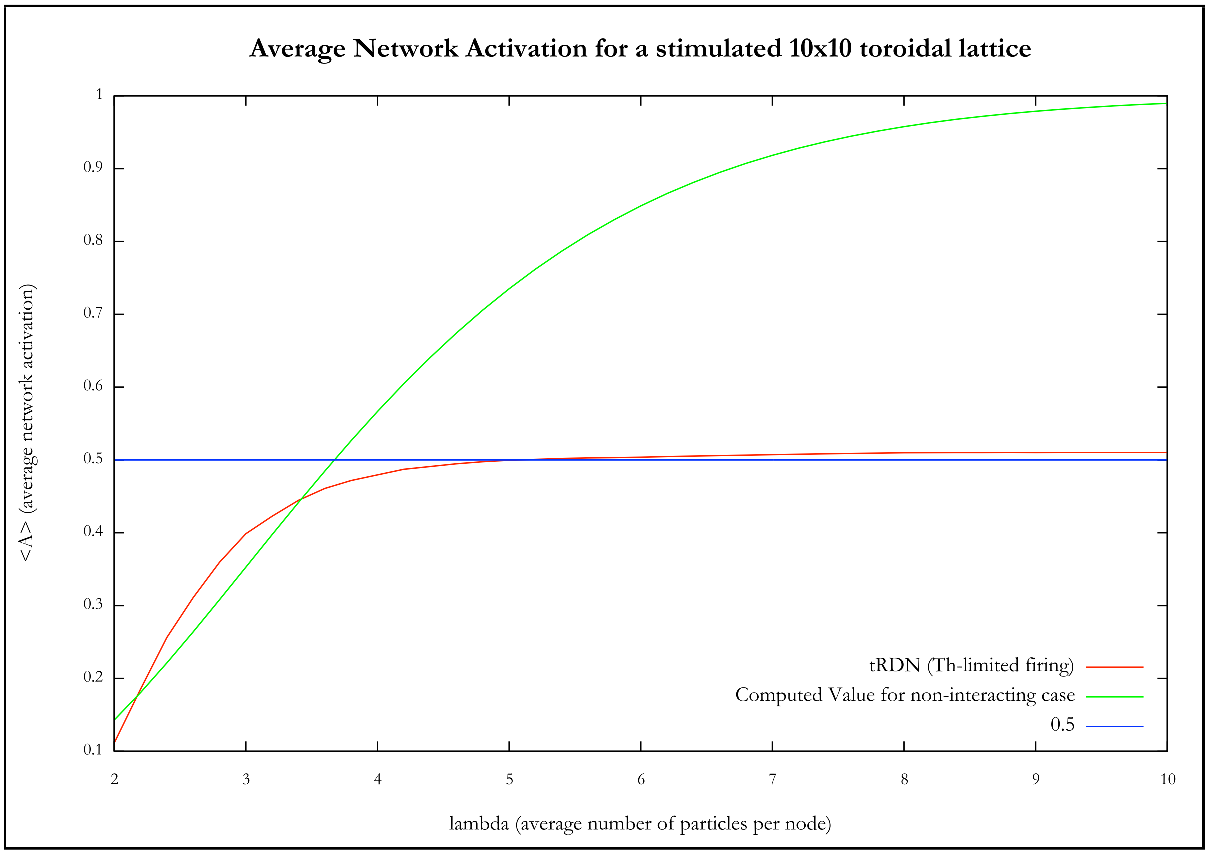


Figure 4.2: Average Network activation $\langle \mathcal{A}_4 \rangle$ for a stimulated 10x10 toroidal lattice. Parameters used are: $d = 0.05$, SR vary. In the x axis is reported the average number of particles per node produced by different SR. Three curves are shown: the first is obtained from a *fire-all* tRDN ($\theta = 4$), while the second is the expected value computed using the expression (4.1). The blue line indicates the value of $\langle \mathcal{A}_4 \rangle = 0.5$. Average values obtained from $5 \cdot 10^4$ time-steps, in a relaxed condition.

Figure 4.4 shows the average Network activation $\langle \mathcal{A}_4 \rangle$ for lower value of λ, computed with $d = 0.005$. Even if around $\lambda = 2$ the curve seemed approaching faster to 0 in the tRDN case than in the Random Walk case, the behaviour is different.

This is due to the fact that in the tRDNs it is more possible to have nodes with more than θ particles because of the threshold itself, that allows the nodes to charge. In the Random Walk, the continuous distribution of particles among the nodes reduces notably this probability.

Figure 4.5 shows the curve of $\langle \mathcal{A}_4 \rangle$ for a threshold-firing tRDN with a larger range of λ, compared to the expected values for a Random Walk on the same graph.

Case: 11 x 11 uniform toroidal lattice

In this set up the chessboard effect is highly reduced, as it can be seen by the third curve in Figure 4.6, because of the uneven number of nodes on the lattice's edges. The mechanism that produces the chessboard-like distribution is inhibited thus, but not totally suppressed: we can have small local chessboard-like zones.

Also in this case the Random Walk (blue line in Fig. 4.6.) reproduces the expected values, while the stimulated tRDN differ, especially for very low and very high values of λ.

4.5.3 Avalanches and fluctuations measures

As we have seen in *SandPile* models, the interesting events occurring are *avalanches.* As we have seen previously, even if the average activation of a tRDN differs from its stimulated RDN counterpart, this does not give us any information about those critical events.

At this point, we must define what an avalanche is.

In the SandPile model an avalanche was considered as a toppling; at a certain point the toppling started and a certain number of sites topples; such a number of sites constitute the *dimension* of the avalanche. In these models it is possible to evaluate the dimension of the avalanche because the toppling time is much smaller than the interval that occurs from one stimulus to the other one: indeed, while a toppling is occurring, it is not possible to add any grain of sand to the pile, but it is necessary to wait that the system has relaxed.

In the stimulated tRDN model, on the contrary, the stimulus is continuous in time. In this way, it is possible that while an avalanche is occurring, there are also a certain number of local topplings occurring. The activation is then filled by a noise due to the stimulus.

We need then to define what in such a system an *avalanche* is.

An avalanche can basically product two effects: on one hand, it can activate in a very small time a very large number of nodes; on the other, it can also activate a restricted number of nodes for a very long time, propagating. In both cases we must have that $\mathcal{A}_\theta(t) > \langle \mathcal{A}_\theta(t) \rangle$; but this condition is not sufficient, because even the fluctuations due to the stimulus brings the activation over its average value.

We can then set a *limit*, ℓ for $\mathcal{A}_\theta(t) - \langle \mathcal{A}_\theta \rangle$, in order to state that any value of $\mathcal{A}_\theta(t) > \langle \mathcal{A}_\theta \rangle + \ell$ constitutes a *big fluctuation*, or an *improbable event*, or *the beginning of an avalanche.*

Every time we have $\mathcal{A}_\theta(t) > \langle \mathcal{A}_\theta \rangle + \ell$, we say we have a **peak**, labeled i, $\mathcal{P}_i$.

Such events are not forbidden in Random Walks, but they are very improbable.

What we expect from a stimulated tRDN these critical events occur with a higher frequency.

A reasonable choice for ℓ is to set it as a multiple of the *standard deviation* of $\mathcal{A}_\theta(t)$, $\sigma_{\mathcal{A}_\theta}$.

I setted the limit to be $\ell = 2.5\ \sigma_{\mathcal{A}_\theta}$, since it has worked well in the following data analysis. It is a reasonable limit, since we expect in the linear case that the mean activation has gaussian-like distribution, and in this case it is unlikely to have $\mathcal{A}_\theta(t) > \langle \mathcal{A}_\theta \rangle\ 2.5\ \sigma_{\mathcal{A}_\theta}$

The sole computation of the global time spent over the *limit* for the activation function, however, doesn't give us a measure of the dimension of the avalanche. Since what we are searching fore are big avalanche, we need to compute other quantities to enhance this property.

I evaluated then the distribution of two quantities: the *consecutive time $\mathcal{A}_\theta(t)$ spends over the limit* and the *integral subtended it.*

Consecutive time over the limit

With this measure I calculate the frequency of the system to have $\mathcal{A}_\theta(t) > \langle \mathcal{A}_\theta \rangle + \ell$ for a certain consecutive time.

This measure express the idea that an avalanche must, in any case, generate fluctuation in the activation that is much larger than the average fluctuation, persisting for a reasonable long time (i.e. the avalanche propagates).

For the i-th peak $\mathcal{P}_i$, which surpasses the given limit at time $t_{i,enter}$ up to time $t_{i,exit}$ we define then

$$\mathrm{t}_\ell(\mathcal{P}_i) := t_{i,exit} - t_{i,enter}$$

I expect to register, in a stimulated tRDN, more persisting fluctuations than in a Random Walk.

Integral subtended to $\mathcal{A}_4(t)$ and the limit

With this measure I want to esteem the dimension of the avalanche. I calculate the area subtended by the activation of the network and limited by ℓ, as:

$$\mathbf{I}_\ell(i) = \int_{t_{i,enter}}^{t_{i,exit}} \mathcal{A}_\theta(t) - (\langle \mathcal{A}_\theta \rangle + \ell)$$

This measure express the idea that an avalanche must, activate for a certain time a certain (large) number of nodes. In this way, high and narrow peaks have the same weight of less high, but larger, peaks.

Also in this case, I expect to have a larger probability to activate many nodes in a tRDN rather than in a Random Walk.

Hitting times

In Markov Chains [15] a **Hitting Time** of a subset I T_I of the State Space $\{X_n\}_n$ is defined as

$$T_I \ := \ inf\{\ n \geq 0 \ : \ X_n \in I\}.$$

it is in practice the minimum time in which the system reaches a certain state in the subset I.

We can then consider the subset $I \ = \ \{\vec{\mathbf{s}}(t) \mid \mathcal{A}_\theta(\vec{\mathbf{s}}(t)) \ > \ \langle \mathcal{A}_\theta \rangle + \ell\}$

and evaluate the distribution of the hitting times of such set; in particular, I analyse the difference $t_{j,enter} - t_{i,enter}$ where $\mathcal{P}_i$ and $\mathcal{P}_j$ are two *consecutive* peaks ($t_{j,enter} \ > \ t_{i,enter}$):

$$\mathfrak{h}_\ell(j) \ := \ t_{j,enter} - t_{j-1,enter} \qquad \text{with } \mathcal{P}_j \text{ immediatly consecutive to } \mathcal{P}_{j-1}$$

4.5.4 Avalanches and fluctuations analysis

I report here the results obtained from simulations ran using *ROnDINE* simulating 2D lattice tRDNs and the corresponding Random Walks.

I focussed in analysing the *threshold firing* case, in order to avoid effects such as the *chessboard*-distribution shown in Sec. 4.5.2, due to the high regularity of the system analysed, being it a small lattice.

As noted in section 4.5.2, the most interesting window to analyse is defined by $\lambda = \langle P_n \rangle / N \ <\sim 2.5$. If $\lambda \simeq 0$, the activity of the network is almost 0, and the small variations are only due to local firing; this regime is not particularly interesting.

I analyse then simulation performed using $1 \leq \lambda \leq 3$. In this range, I expect to see a notably difference in the fluctuations behaviour than from the stimulated tRDN and the respective Random Walk.

Case: 10 x 10 uniform toroidal lattice

I report here the results obtained from the above mentioned measures $\mathfrak{t}_\ell(\mathcal{P}_i)$, $\mathbf{I}_\ell(i)$ and $\mathfrak{h}_\ell(j)$, in a stimulated 10 x 10 uniform toroidal lattice, obtained from a series of simulations performed with *ROnDINE*.

As in Sec. 4.5.2, the mean values and the histograms are computed on 50000 time-steps, after other 50000 time steps. Histograms then represents the probability of events of a certain dimension (it depends from the measure done). Networks have dissipation $d = 0.005$ and source-rate varying, in order to have different values for λ.

The possibility to have an *avalanche* is then represented by the height of the columns in the tails of the histograms.

Figure 4.7 reports the distribution of $\mathbf{I}_\ell(i)$ obtained for a *threshold-firing* tRDN and the respective Random Walk. Results for $\lambda = 2$ and $\lambda = 2.4$ are reported.

As it can be seen, in the case of the tRDN with $\lambda = 2$ the probability of a large event is much larger than in the case of a Random Walk. Also with $\lambda = 2.4$ we can note the difference of probability to have such events, but it is notably smaller than in the previous case: we are very close to the change of regime noted in Sec. 4.5.2 for $\lambda > 2.5$.

Figure 4.8 shows the distribution obtained for $\mathfrak{t}_\ell(\mathcal{P}_i)$ and Figure 4.9 the distribution for the hitting times $\mathfrak{h}_\ell(j)$. These Figures confirms what has been said above.

Case: 100 x 100 uniform toroidal lattice

I replicated the same experiment by increasing the number of nodes in the lattice, from $N = 100$ to $N = 10^4$, by using a 100 x 100 toroidal lattice.

Mean values and the histograms are computed on 50000 time-steps, after other 50000 time steps. Networks have dissipation $d = 5 \cdot 10^{-4}$ and source-rate varying, in order to have different values for λ.

Figure **??** reports the distribution of $\mathbf{I}_\ell(i)$ obtained for a *threshold-firing* tRDN and the respective Random Walk. Results for $\lambda = 2$ and $\lambda = 2.4$ are reported.

Figure 4.11 shows the distribution obtained for $\mathfrak{t}_\ell(\mathcal{P}_i)$.

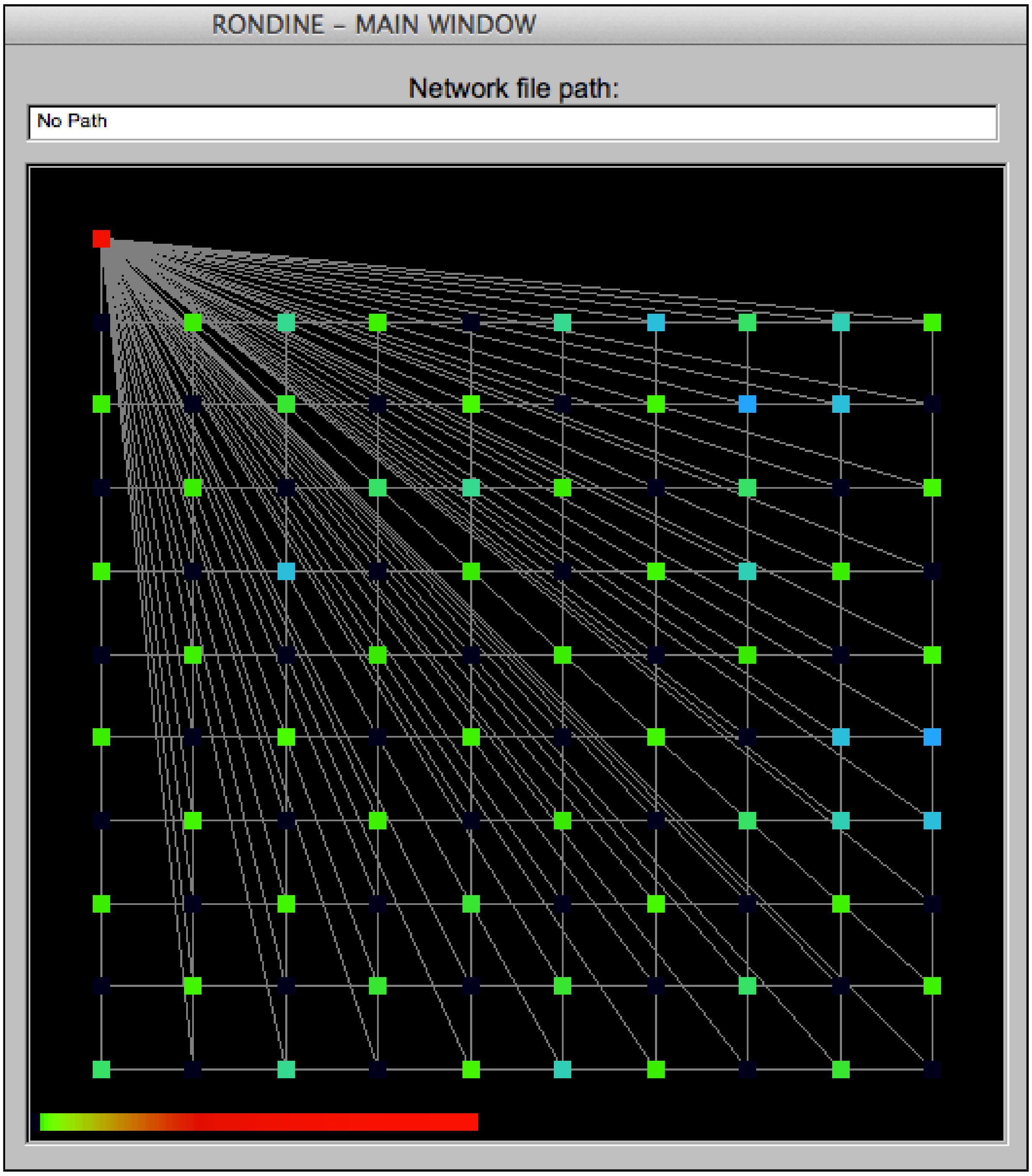


Figure 4.3: Screen shot of *ROnDINE* showing the chessboard-like distribution of particles among node, on a stimulated 10 x 10 toroidal lattice. On the black background, the coloured squares represents the nodes, while the grey lines represent the links connecting them. Each node has a saturation level: the hotter is the colour, the more populated is the node (see Appendix A); where there is a black square, we have an *empty node*. On the top you can see a red node, which is the reservoir - in fact, it is connected to all the other nodes throughout the dissipative links. The other nodes represent the nodes of the lattice, and, as you can see, they are mostly occupied by particles following a chessboard-like scheme. In this care the mean activation is a bit larger than 0.5.

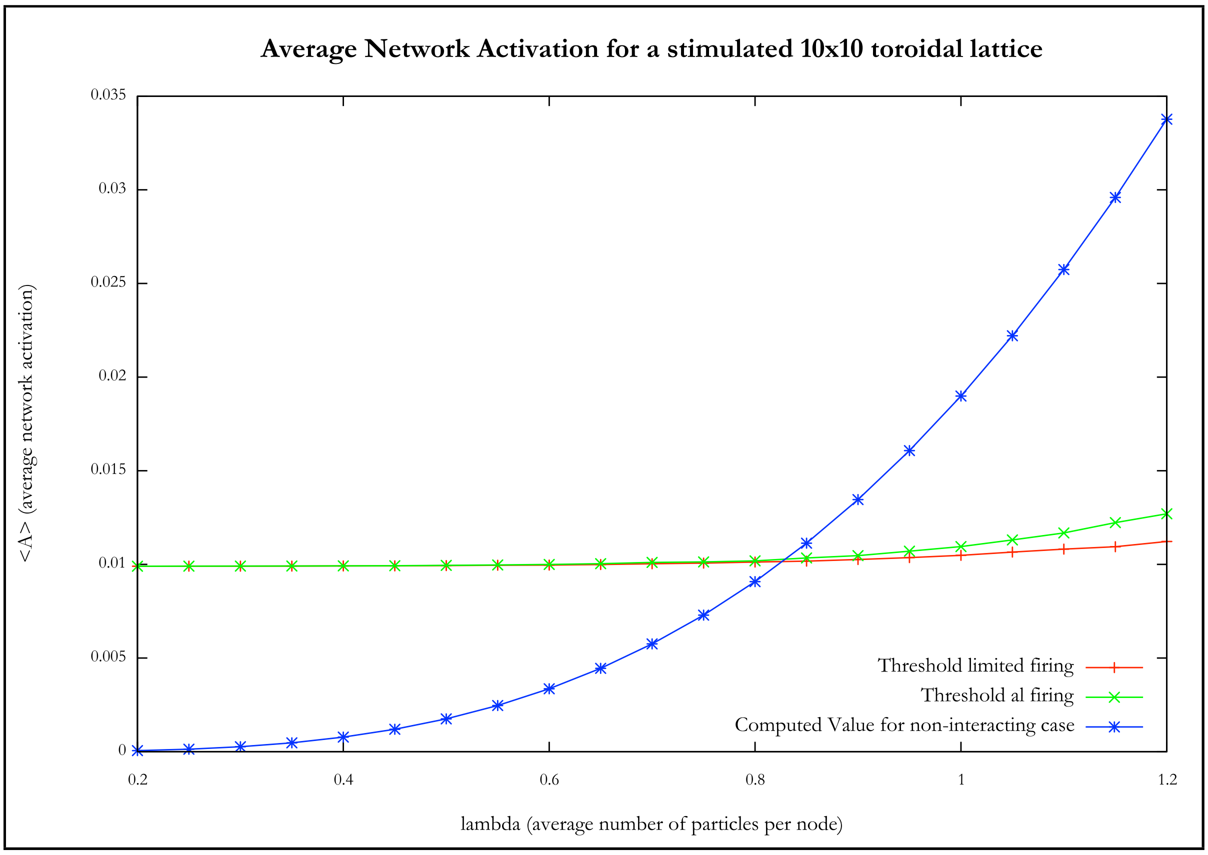


Figure 4.4: Average Network activation $\langle \mathcal{A}_4 \rangle$ for a stimulated 10x10 toroidal lattice. Parameters used are: $d = 0.05$, SR vary. The condition are equal to the prefious case (see Figure 4.1), but with lower values of SR producing lower values of Λ. The different behaviour for $\lambda \rightarrow 0$ is caused by the possibility, for the nodes in a tRDN, to charge.

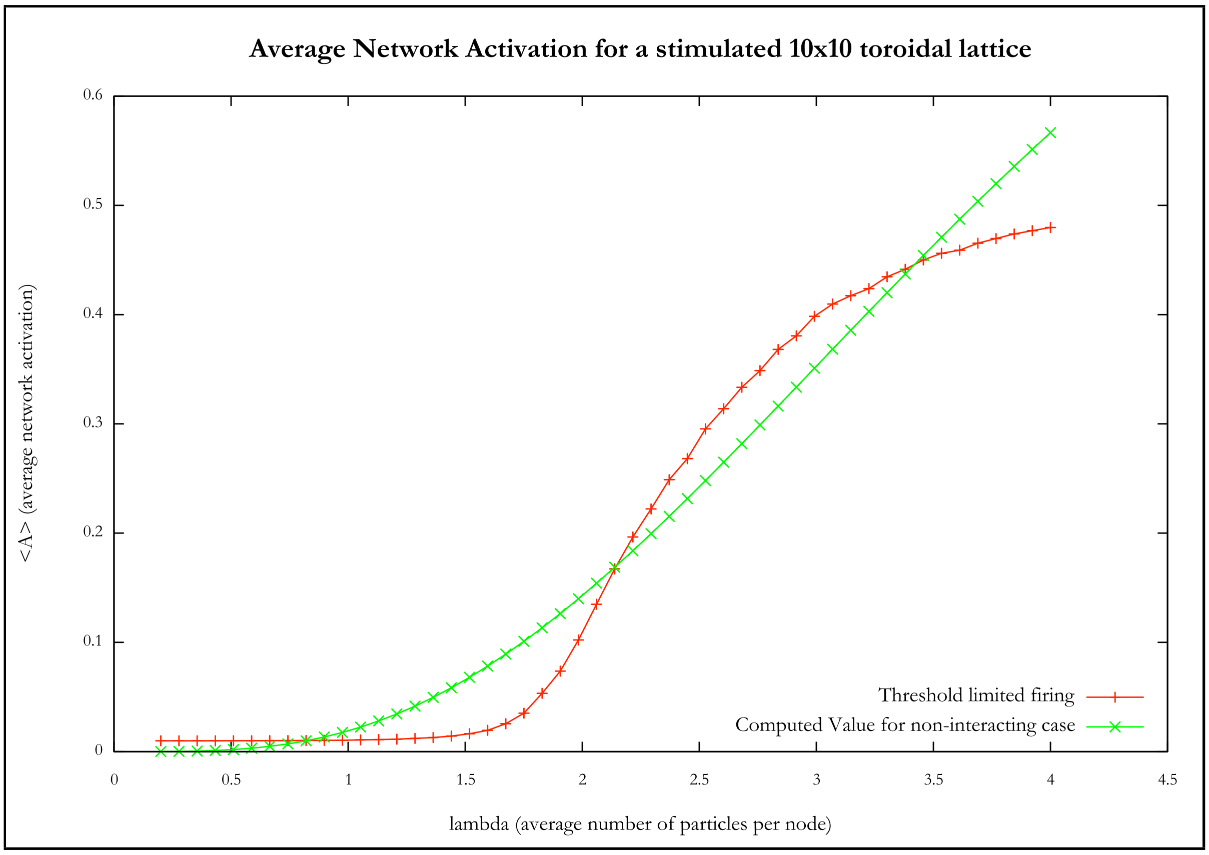


Figure 4.5: Average Network activation $\langle \mathcal{A}_4 \rangle$ for a stimulated 10x10 toroidal lattice. Parameters used are: $d = 0.05$, SR vary. The condition are equal to the previous case (see Figure 4.1), with a wider range for λ

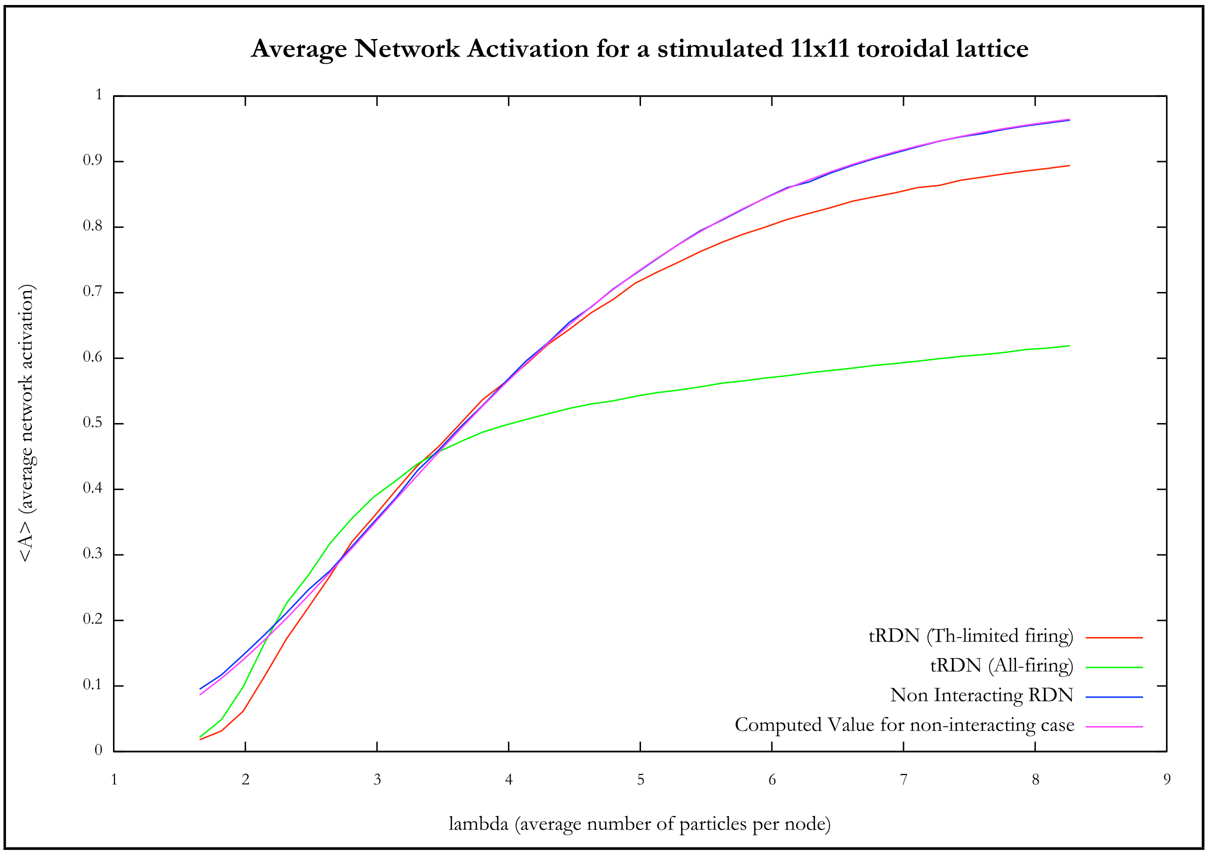


Figure 4.6: Average Network activation $\langle \mathcal{A}_4 \rangle$ for a stimulated 11x11 toroidal lattice. Parameters used are: $d = 0.05$, SR vary. In the x axis is reported the average number of particles per node produced by different SR. Four curves are shown: the first (red) is obtained from a *threshold-limited* tRDN ($\theta = 4$), the second (green) from a *fire-all* tRDN in the same set up, the third (blue) is obtained from a Random Walk on this network while the last one (purple) is the expected value computed using the expression (4.1). Average values obtained from $5 \cdot 10^4$ time-steps, in a relaxed condition.

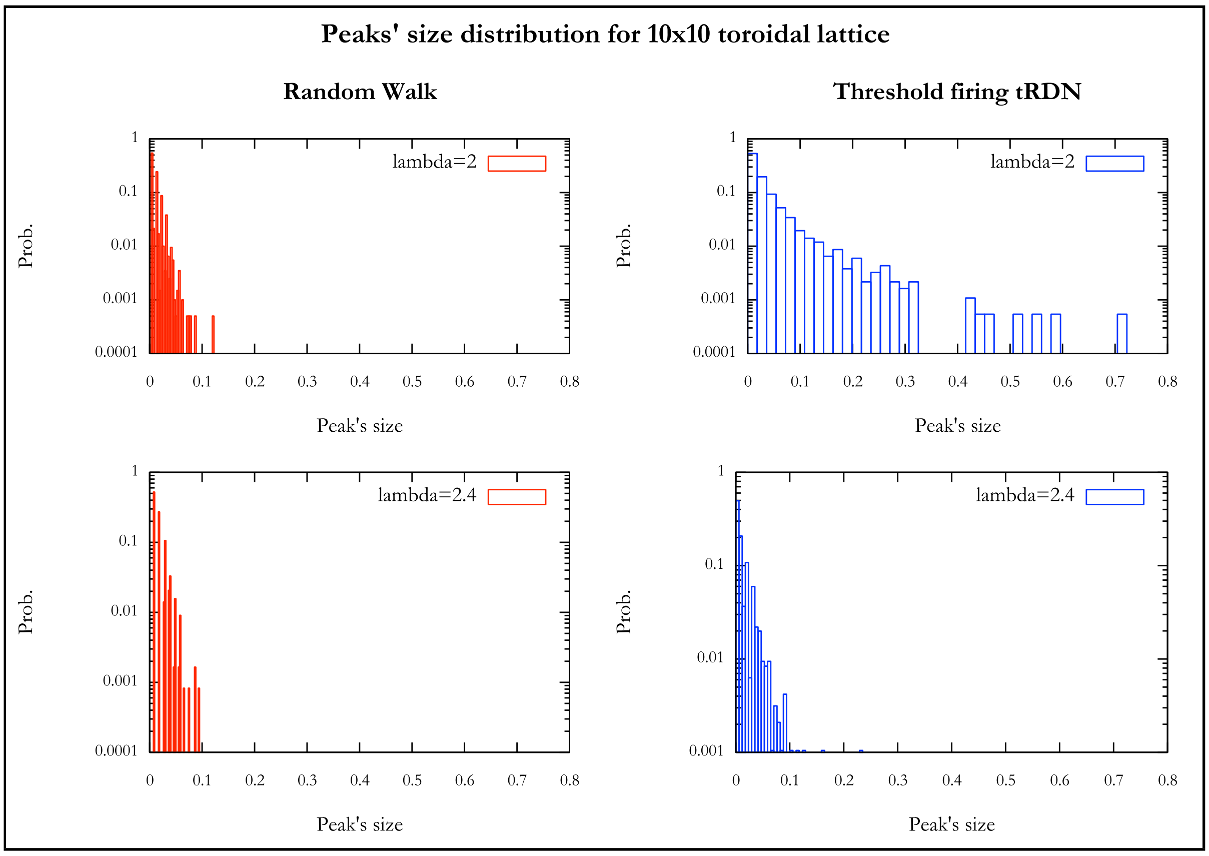


Figure 4.7: Distribution of $\mathbf{I}_\ell(i)$, $\ell = 2.5\,\sigma_{\mathcal{A}_\theta}$, for a 10 x 10 uniform toroidal lattice with dissipation $d = 0.005$ and stimulus varying, in order to vary λ (average number of particles per node). In the first row of the image we have histograms obtained for networks having $\lambda = 2$, in the second row $\lambda = 2.4$; the first column reports data obtained from a Random Walk, while the second column shows the data obtained from a threshold-limited tRDN. As it can be seen, the distributions in second column shows fatter tails, and in the second row this tails reduced due to the change of regime noted in Sec. Sec. 4.5.2 for $\lambda > 2.5$.

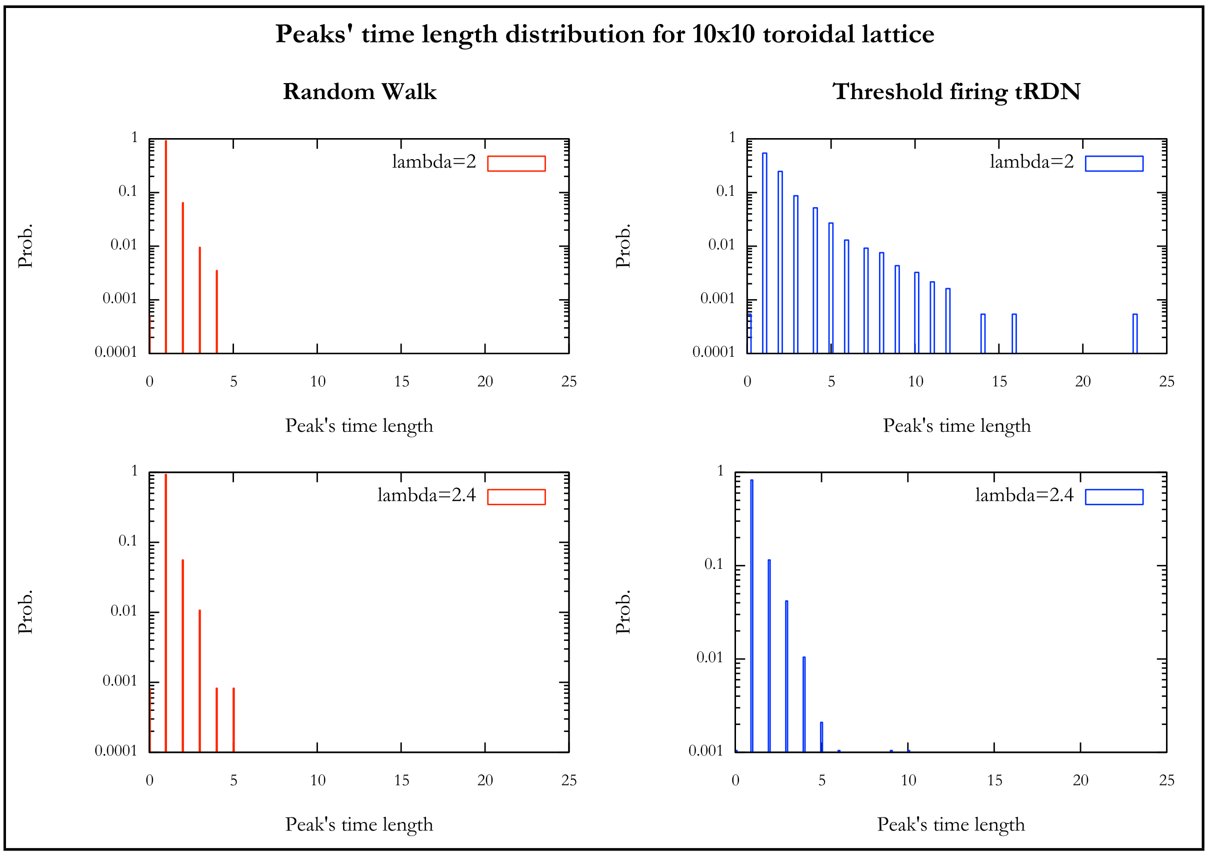


Figure 4.8: Distribution of $\mathrm{t}_\ell(\mathcal{P}_i)$, $\ell = 2.5\,\sigma_{\mathcal{A}_\theta}$, for a 10 x 10 uniform toroidal lattice wuth dissipation $d = 0.005$ and stimulus varying, in order to vary λ (average number of particles per node). In the first row of the image we have histograms obtained for networks having $\lambda = 2$, in the second row $\lambda = 2.4$; the first column reports data obtained from a Random Walk, while the second column shows the data obtained from a threshold-limited tRDN. As it can be seen, the distributions in second column shows fatter tails, and in the second row this tails reduced due to the change of regime noted in Sec. Sec. 4.5.2 for $\lambda > 2.5$.

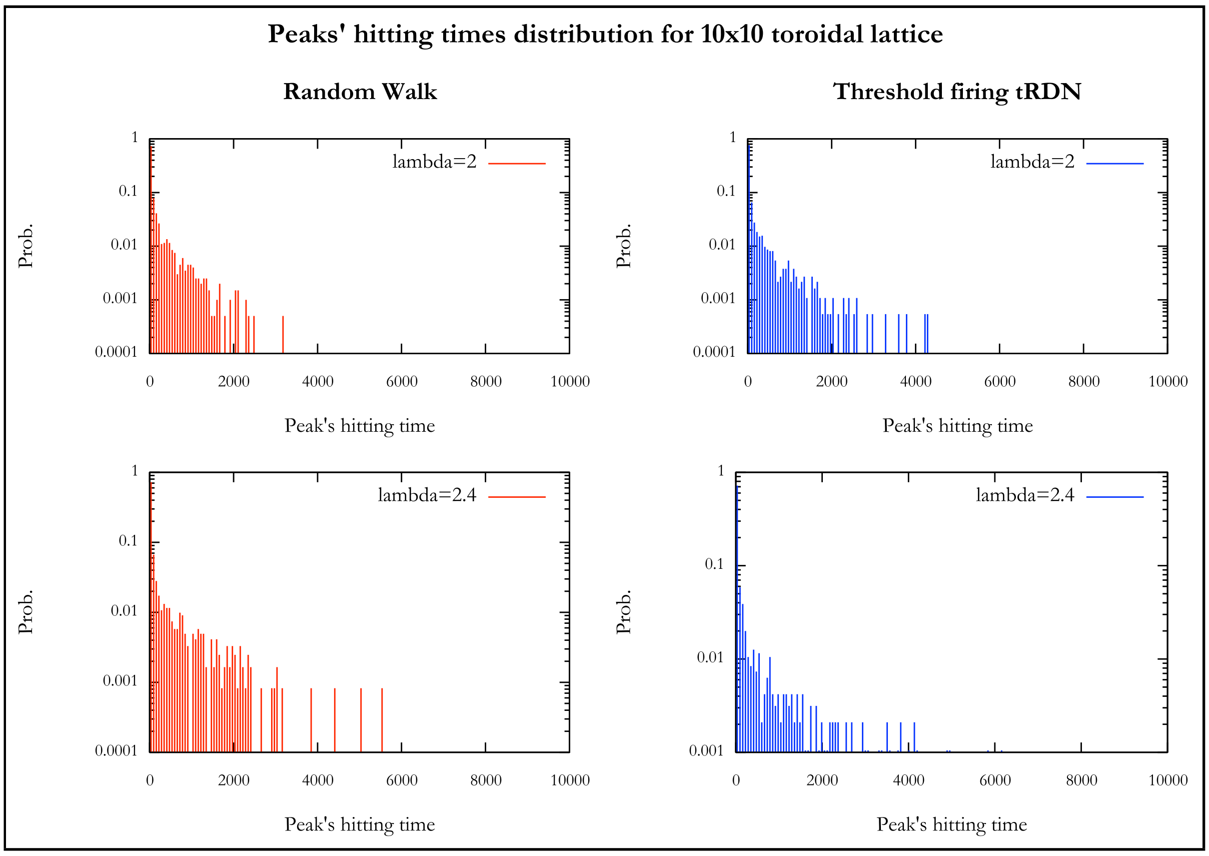


Figure 4.9: Distribution of $\mathfrak{h}_\ell(j)$, $\ell = 2.5\,\sigma_{\mathcal{A}_\theta}$, for a 10 x 10 uniform toroidal lattice with dissipation $d = 0.005$ and stimulus varying, in order to vary λ (average number of particles per node). In the first row of the image we have histograms obtained for networks having $\lambda = 2$, in the second row $\lambda = 2.4$; the first column reports data obtained from a Random Walk, while the second column shows the data obtained from a threshold-limited tRDN. As it can be seen, the distributions in second column shows fatter tails, and in the second row this tails reduced due to the change of regime noted in Sec. Sec. 4.5.2 for $\lambda > 2.5$.

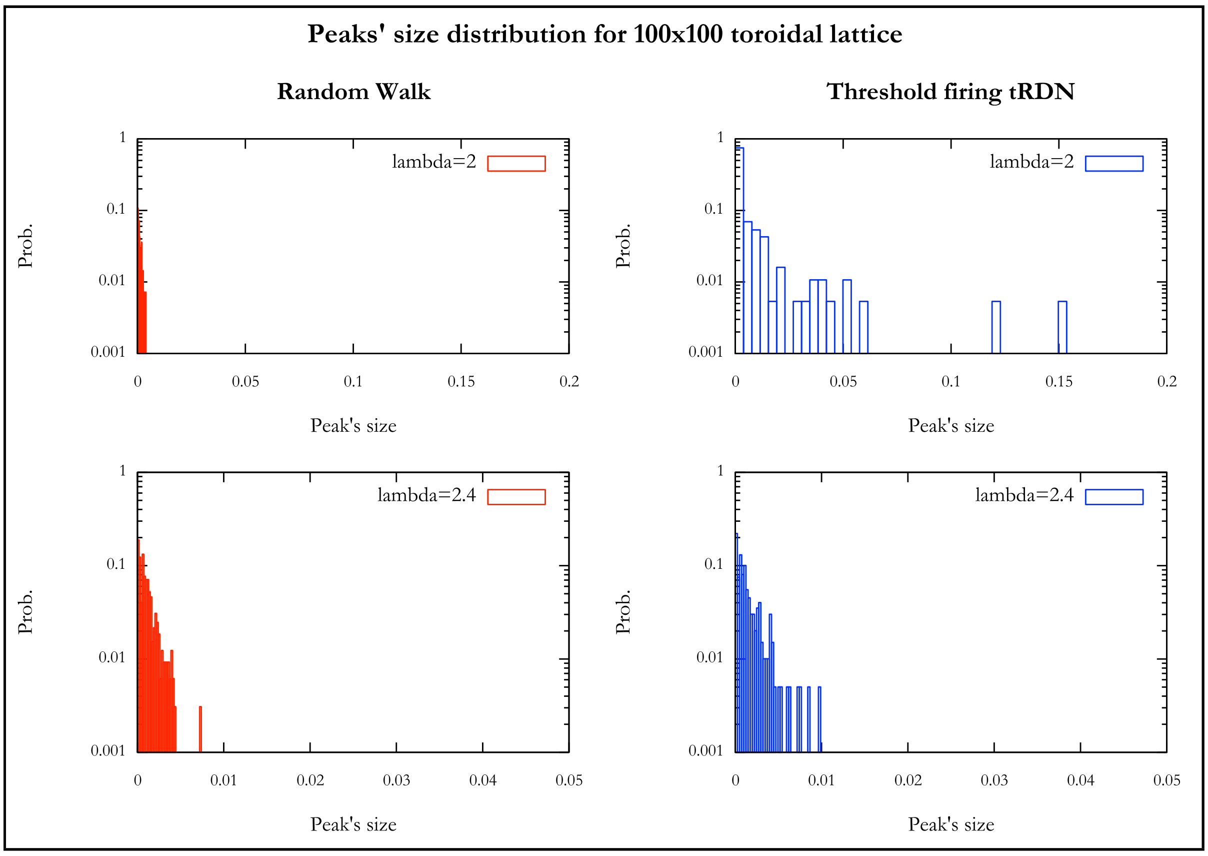


Figure 4.10: Distribution of $\mathbf{I}_\ell(i)$, $\ell = 2.5\,\sigma_{\mathcal{A}_\theta}$, for a 10 x 10 uniform toroidal lattice with dissipation $d = 0.005$ and stimulus varying, in order to vary λ (average number of particles per node). In the first row of the image we have histograms obtained for networks having $\lambda = 2$, in the second row $\lambda = 2.4$; the first column reports data obtained from a Random Walk, while the second column shows the data obtained from a threshold-limited tRDN. As it can be seen, the distributions in second column shows fatter tails, and in the second row this tails reduced due to the change of regime noted in Sec. Sec. 4.5.2 for $\lambda > 2.5$.

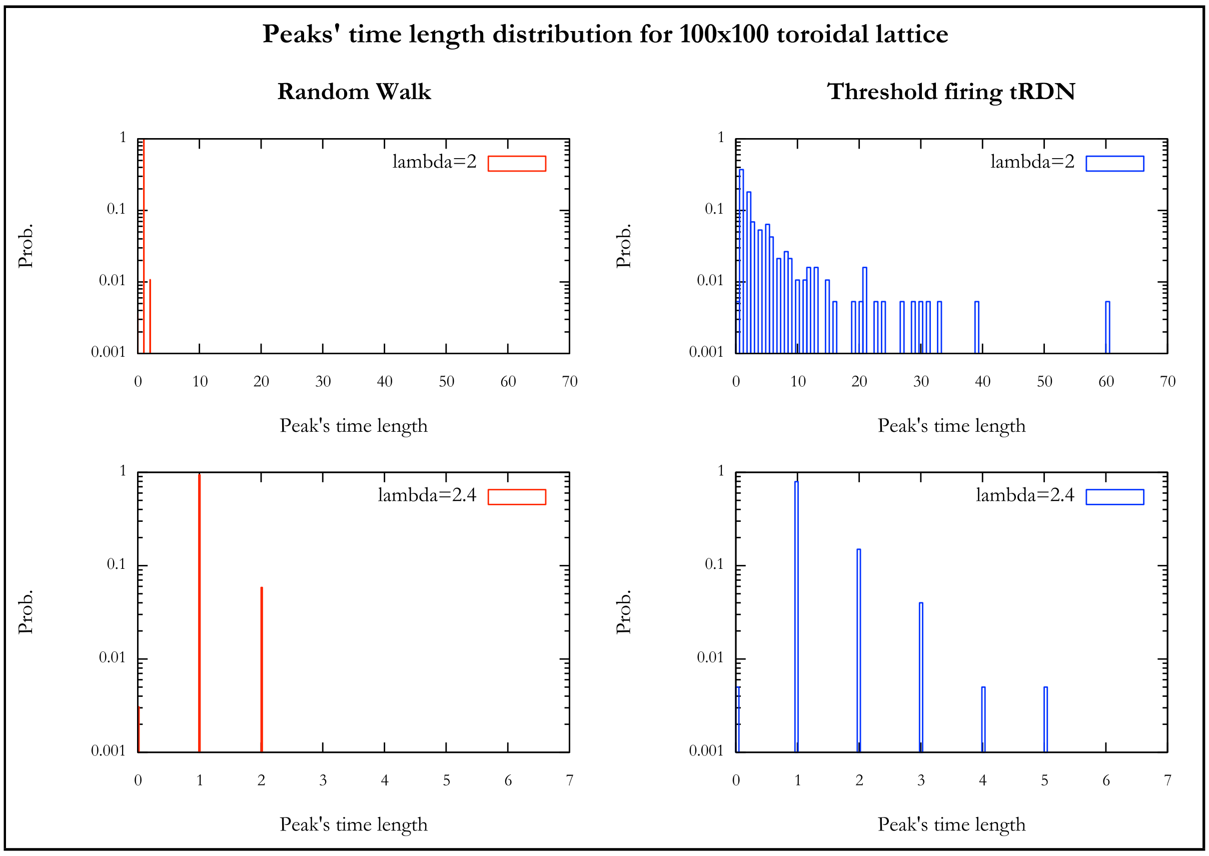


Figure 4.11: Distribution of $\mathfrak{t}_\ell(\mathcal{P}_i)$, $\ell = 2.5\,\sigma_{\mathcal{A}_\theta}$, for a 10 x 10 uniform toroidal lattice wuth dissipation $d = 0.005$ and stimulus varying, in order to vary λ (average number of particles per node). In the first row of the image we have histograms obtained for networks having $\lambda = 2$, in the second row $\lambda = 2.4$; the first column reports data obtained from a Random Walk, while the second column shows the data obtained from a threshold-limited tRDN. As it can be seen, the distributions in second column shows fatter tails, and in the second row this tails reduced due to the change of regime noted in Sec. Sec. 4.5.2 for $\lambda > 2.5$.

Chapter 5

Conclusions

A series of models for Stochastic Random Dynamics have been studied, focussing on their fluctuations' properties, which have been individuated with proper measures for critical events.

In particular Random Dynamical Networks have been defined as Markov Chains on graphs: in the analogy used, the states corresponded to a node of the graph and the transition rates were given by the weights matrix of the graph.

In parallel, an original C++ program has been developed, *ROnDINE* (described in Appendix A) which has been used to perform the simulations needed and to have a visualization of the simulated Random Dynamical Networks.

Initially it has been defined a Random Walk on a Network as the movement of a single particle on a graph, with discrete time; its average behaviour, which also defines the collective behaviour of M non-interacting particles moving on the same graph, has been studied.

With the introduction of a *reservoir* in the network by adding one node to the graph and connecting each node to it, it has been possible define a *dissipation* term on the system. The reservoir node worked as a *stimulus* for the system too, by injecting particles to randomly chosen nodes with a source rate SR. The average number of particles in the network was therefore given by SR/d.

It has been then defined a simple *Integrate and Fire model* for Neural Networks, based on the Random Dynamical Network model. Such model are inspired by both the Integrate and Fire neuronal models an the SandPile models, whose main feature is the *threshold*, θ, that allows the particles to escape a node i only if that node has a number of particles $\mathbf{s}_i \geq \theta$.

I named these models tRDNs (threshold Random Dynamical Networks), and I defined two variants: *all firing* and *threshold firing* (or *threshold-limited firing*). In the first model, a node with a number of particles equal or grater than the threshold fires *all* its particles in a single time step; in the second model such a node fire *only* θ particles. Each model has a *reservoir*, with a dissipation rate d and a source rate SR. In this way, I study *stimulated* tRDNs.

These models are small and simple Neural Networks, and they are statical (i.e. there is no learning): the weights of the network and the connections does not change in time. The aim of these models is to analyse the behaviour of IF neurons on given networks' structure, in order to understand better their dynamics and the relationships with the graph's structure. This model can describe both very simple and very complicated neural networks, depending on the graphs we chose. The comparison with Random Walks networks on the same graphs can give interesting informations on the model's behaviour.

Simulations done with the program ROnDINE showed how in particularly regular networks, such as small toroidal 2D $N \times N$ lattices with N even, a tRDN favourites *chessboard*-like states. This family of states is characterized by a particular configuration that, even if it can be artificially created in a Random Walk network, is very uncommon. Once such a state is reached it is very difficult for the systems to switch to another one . The *threshold-limited* tRDN inhibits this behaviour, by reducing the probability for a node to be empty.

The formation of critical events, or *avalanches*, has been studied in a tRDN on very simple networks, such as 2D lattices, in order to understand the model's behaviour. More precisely, I defined an avalanche as the activation of a notable number of nodes, where an activated node is a node with at least θ particles and the *network activation* at time t given the threshold θ is the mean activation of each node $\mathcal{A}_\theta(t)$. The confrontation of this quantity in the case of a tRDN and the respective Random Walk has provided interesting results: given that the mean network activation $\langle\mathcal{A}_\theta\rangle$ of a Random Walk can be derived from a Poissonian distribution, it has been possible to compare theoretical values to simulations' values.

The analysis of $\langle\mathcal{A}_\theta\rangle$ in three cases (all firing tRDN, threshold firing tRDN and a Random Walk on the same network), allowed to identify $\lambda = SR/d$ as a leading tuning parameter that defines the dynamics of the network. For $\lambda \leq 1 \leq 3$, the behaviour of a tRDN appeared notably different from the behaviour of a Random Walk.

An *activation peak* has therefore been defined, with respect to "limit" ℓ setted as $\ell = 2.5\sigma_{\mathcal{A}_\theta}$, and it became the centre of the investigations. Three measures have been then defined: the length of a peak of $\mathcal{A}_\theta(t)$, the integral of a peak and the hitting time between two peaks ($\mathfrak{t}_\ell(\mathcal{P}_i)$, $\mathbf{I}_\ell(i)$ and $\mathfrak{h}_\ell(j)$.).

I studied then the distributions of these measures, expecting fatter tails in those derived from a simulation of a tRDN, which are the signal of an avalanche.

The simulations showed in these distributions the results I expected, by confirming that I focussed the attention in an interesting range for the tuning parameter λ.

The results I obtained from this work gives a background for further investigations on the tRDN models. Such models, being a mixture of Integrate and Fire neuron models and SandPile model, can give interesting informations about the behaviour of complex systems, such as neural networks. By replicating the parallelism between the 2D lattices tRND and the 2D SandPile model to more complicated networks and Abelian Sand Pile models it could be possible to characterize more precisely the tuning parameters λ, SR and d.

An interesting choice for further investigations is represented by Artificial Clustered Network (see Sec. 3.3.1) because of their nested structure and the possibility to define properly two evolution time scales: in such a situation, we can expect also two scales for the avalanche events, which can be tuned in order to obtain different effects.

Appendix A

ROnDINE

ROnDINE (which stay for RandOm DynamIcal NEtwoks) is a C++ software for Discrete Stochastic Dynamics on Networks Simulations. It simulates a RDN (see Chapter 2), performs statistical analysis of the evolving system and gives a real-time visualizations of their states. **ROnDINE** needs two third-party libraries:

- **igraph 0.7** [12] (*http://igraph.org/*) - a library for graphs' data structures, operations, and algorithms.
- **FLTK 1.3** [13] (*http://www.fltk.org/*) - a graphic library that includes OpenGL library.

ROnDINE's project name is *Laura*, in honour of *Laura Bassi*[1], and its repository can be found at gitHUB https://github.com/ETRu/Laura).

[1]from Wikipedia: " Laura Maria Caterina Bassi (31 October 1711 - 20 February 1778) was an Italian physicist and academic, recognized as the first woman in the world to earn a university chair in a scientific field of studies. She received a doctoral degree from the University of Bologna in May 1732, the third academic qualification ever bestowed on a woman by a university, and the first woman to earn a professorship in physics at a university in Europe. She was the first woman to be offered an official teaching position at a university in Europe." (`https://en.wikipedia.org/wiki/Laura_Bassi` , retrieved 25 Nov 2015)

A.1 Installation an General Description

A.1.1 Compatibility

ROnDINE has been developed using *OSX 1.9.5 (Mavericks)*, and tested on *Ubuntu 10.4*, *Kubuntu 14* and *Fedora 21.* There is no specific reason suggesting its incompatibility with other OSX systems and LINUX Intel systems, provided that igraph and fllt libraries are correctly installed and linked; however, its perfect functioning cannot be guaranteed.

A.1.2 Installation

To install ROnDINE it is necessary to install the third party libraries igraph 0.7 and fltk 1.3 . Please note that igraph 0.7 requires *libxml* and fltk 1.3 requires *libX11.*

When installing fltk, please check if OpenGL is enabled. In doubt, configure fltk's installation using *./configure -enable-gl* .

Once the libraries are installed, just run the *make* command in ROnDINE's src folder. If it doesn't work, check in the makefile and if necessary change If everything works, to run ROnDINE run the exectuable named "RONDINE", e.g. by typing *./RONDINE* .

What does ROnDINE do?

Basically, ROnDINE simulates the evolution of a system composed by N particles moving on a graph, with Markovian dynamics induced by the Weights Matrix of the graph (see section 2.0.3), showing to you what is happening and computing a series of measure.

A.2 ROnDINE's engine

The core engine of ROnDINE is constituted by the *run()* and *turbo(run)* routines. The first one runs the simulation step by step, passing to the graphic interface the data needed to draw the RDN and the graphs in the data window, while *turborun()* runs a long simulation without interacting with the interface: this allow *turborun()* to be much f aster than *run()*, renouncing to graphic visualization.

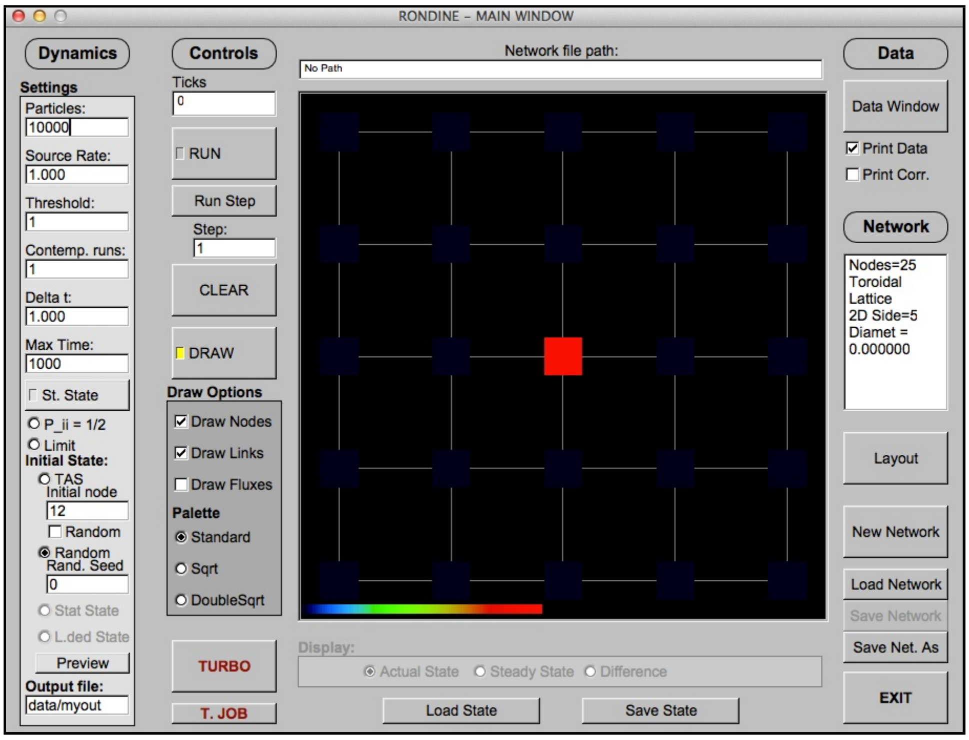
Both the routines rely on the same algorithm for the computation of the new state of the network. In a single time step Δt, for each non empty node i and for each particle of the node allowed to move (see Sec.s 1.6 and 1.6.1), the arrival node of the particle is computed using a *spinning wheel* system.

Each node j connected to node i has a transition probability $\mathbf{M}_{ji}$, given by the Weight's Matrix of the network. The sum of these probabilities must be 1. We can then assign to each candidate an interval in $[0:1]$. We number the k candidates from 1 to k, and assign to each candidate the respective *cumulative* probability $P(n) = \sum\limits_{n=1}^{k} \mathbf{M}_{ni}$. Then, to each candidate it is assigned the interval defined by $I_n = (P(n-1), P(n)]$, with $I_1 = [0, P(1)]$.

The program then generates a pseudo-random number between 0 and 1 (using a third part generator, MT[41]) and search for the interval where the number belongs: in this way, it selects the arrival node.

A.3 ROnDINE's interface

As you start ROnDINE, you will see this as its two initial windows (Figure A.1 and A.2):



Settings
Particles: 10000
Source Rate: 1.000
Threshold: 1
Contemp. runs: 1
Delta t: 1.000
Max Time: 1000
☐ St. State
○ P_ii = 1/2
○ Limit
Initial State:
○ TAS
Initial node 12
☐ Random
● Random
Rand. Seed 0
○ Stat State
○ L.ded State
Output file: data/myout

Ticks 0
Step: 1
Draw Options
☑ Draw Nodes
☑ Draw Links
☐ Draw Fluxes
Palette
● Standard
○ Sqrt
○ DoubleSqrt

Network file path: No Path
Display: ● Actual State ○ Steady State ○ Difference

☑ Print Data
☐ Print Corr.
Nodes=25
Toroidal
Lattice
2D Side=5
Diamet =
0.000000

Figure A.1: ROnDINE's main window as the program starts.

A.3.1 Main Window

Right in the middle of the **Main Window**, you can see the representation of the network we are working on: the **default network** is a 5×5 *toroidal bidimensional lattice*, i.e. a network of 25 nodes.

Each node is connected to the others according to the *stochastic matrix* $\mathbf{M}$: you can find it printed on the bash. The element $\mathbf{M}_{ij}$ of the stochastic matrix represent the "strength" of the edge connecting node j to node i. More specifically, the elements $\mathbf{M}_{ij}$ represents *transition probabilities* for the generic particles moving on the network. If $\mathbf{M}_{ij} > 0$, the edge connecting j to i is drawn as a straight line.

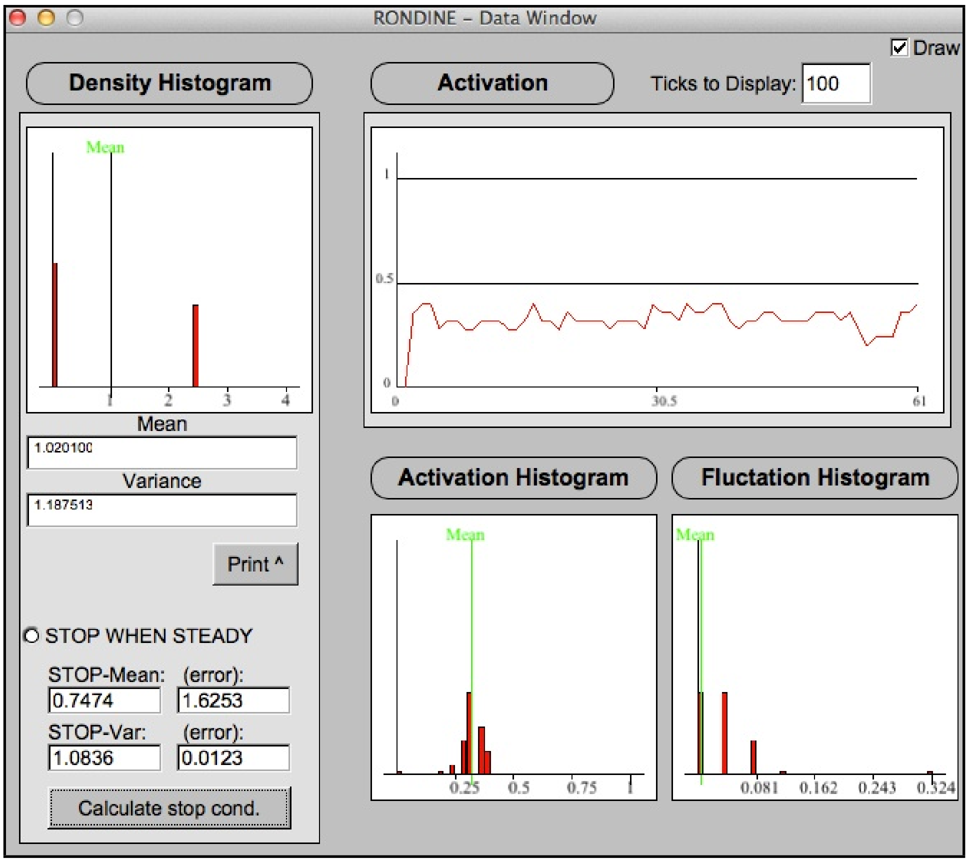
As you can see, as ROnDINE starts the nodes displayed have different coulors: one in red, while the rest are dark-blue.

The *colour* of the nodes represents the *density of particles* in each node. The red node, in the default network, contains *all* the particles of the system.

On the bottom of the black frame, you can see a *palette* (Figure A.3):

the palette shows how the saturation of the displayed nodes changes with respect to the density of particles in the node, from the minimum (dark blue, empty node) to the maximum (pure red, ll the system particles in the node).

On the left of Figure A.3 you can see three different possible palettes: *standard*, *sqrt* and *doublesqrt*. The second and the third options change the standard palette by applying one or two times the *sqrt*


RONDINE – Data Window

☑ Draw

Ticks to Display: 100

Mean: 1.020100

Variance: 1.187513

○ STOP WHEN STEADY

STOP-Mean: 0.7474 (error): 1.6253

STOP-Var: 1.0836 (error): 0.0123

Figure A.2: ROnDINE's data window as the program starts

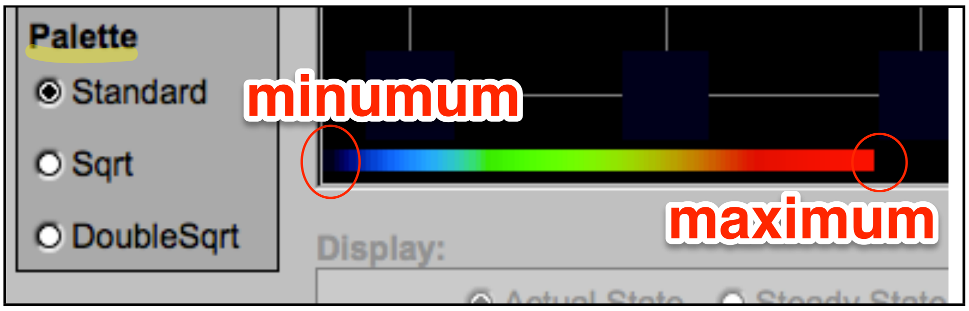


Figure A.3: ROnDINE's palette and palette settings

function at the hue functions. Those palettes are very useful if you work with low-populated nodes, e.g. when you have a network with many nodes, or when some node is notably less accessible than the others. You can change the palette whenever you want.

Now let's have a look to the *left side* of ROnDINE's window.
There are two columns, named *Dynamics* and *Controls*.

Dynamics Settings

The **Dynamics** column allow you to define the dynamical properties of your system. Given a network (the default one, or another network), your particles will move accordingly the transition probabilities given by the stochastic matrix $\hat{M}$, which is fully given by the network. However, there are other parameters of the system, uncorrelated from the network, that you might want to control:

- **Particles** - *the number of total particles moving on the network.*
- **Source Rate** - *if the* ***reservoir*** *node is present (see Sec. 3.4), this value defines SR.*
- **Threshold** - *this value corresponds actually to $\theta - 1$ wrt the tRDN model (see Sec. 1.6). If it is setted to 0, we have a Random Walk.*
- **Contemp. runs** - *the number of contemporary simulations.* When you simulate more than one contemporary runs, ROnDINE will show you the average of all the systems it is running; it will as well print the data obtained from the averages of all the contemporary simulations.
- **Delta t** - *the size of the time step (see Sec. 1.4.5)* (this parameter is useful only if the *P_ii=1/2* option is activated - see below)
- **Max time** - *The maximum number of evolution steps performed*
- **Stat State** - this button *computes the linear stationary state* (LSS) [2] of the system and activates all the functions of ROnDINE subordinated to this computation, e.g. the possibility to display the linear stationary state on the network. Once the LSS has been computed, the button turns yellow. You can disable the LSS by pushing it again: this can be useful if you are not interested in the quantities computed using the LSS, and prefer a faster simulation.
- **P_ii=1/2** - This option *allows the particles to not move from the node where they are.* In fact, *every* stochastic matrix **M** generated by ROnDINE has *null diagonal.* If the P_ii=1/2 option is not checked, the particles are forced to leave the node where they are, accordingly to the transition probabilities given by **M**. If this option is checked, then every particle has probability 1/2 to not move; i.e., for every node i, we can say that $P_{ii} = 1/2$. This does *not* mean that $\mathbf{M}_{ii} = 1/2$, because in this way the stochastic normalization would be broken. The particles decides firstly if it will move or not, and then, according to **M**, it decides *where.* All of this applies if *Delta t* is setted to 1. For different values, the probability to not move of a particle is actually given by $P_{ii} = \Delta t \cdot 1/2$.
- **Limit** - This option, when the threshold is larger than 0, *limits the number of particles leaving a node to theta* (i.e we have a *threshold firing tRDN*).
- **Initial state - TAS** - This round check sets the *Total ASymmetric* initial state for the systems, i.e. a state where all particles are stored in a single node. The id of the node (note: nodes counts from 0 to $N - 1$) can be setted in the input-text immediately below the round check button.
- **Initial state - RAN** - This round check sets the *RANdom* initial state for the system. The N particles are distributed randomly among the network with the same probability for every node. The input-text defines the *seed* for the pseudo-random number generator; in this way, a seed defines one and only one state. If you want to change the RAN initial state (but maintain the distribution), you can just change the seed used.

[2] the *linear stationary state* is here intended as the **first eigenvector** of the stochastic matrix **M**, i.e. the vector of the matrix having eigenvalue 1. See Sec. 3.1

- **Initial State - Stat State** - This round check sets the initial state as the LSS. Since we have discrete particles, the stationary state cannot be exact: this initial state rates generates an initial state by distributing the particles according to the LSS probability distribution. If the LSS is not computed, this round check is deactivated.
- **Preview** - This button *displays the initial state chosen* on the network.
- **Output** - Here you can *define the name of the output files.* By default ROnDINE, when running, generates from 3 to 7 different output files, each named as *myoutNUMBER.txt*, there NUMBER is the index (from 1 to 7) defining the different output files. You can change the name for different simulations (e.g. *newname* will generate *newnameNUMBER.txt* output files). You can also define a path inside teh folder where is the *Project* executable; however, ROnDINE cannot create new folders; if you want to salve your output in a sub-folder you must firstly create it, and then you can give ROnDINE the path to the folder. For example, you can create a folder named *output* in ROnDINE's folder, and then change *myout* with *output/myout.*

Control Settings

The second column allow you to control the time evolution of your system. The button **RUN** runs the simulation: you can immediately see that the left column becomes inactive. If you click on the *RUN* button again, the simulation stops; however, the left column does not activate. When you have stopped a simulation with the button *RUN*, by pushing it again you can **continue** the simulation from the last state displayed. Obviously, you cannot change the simulation settings if you want to *continue* the simulation you just stopped!

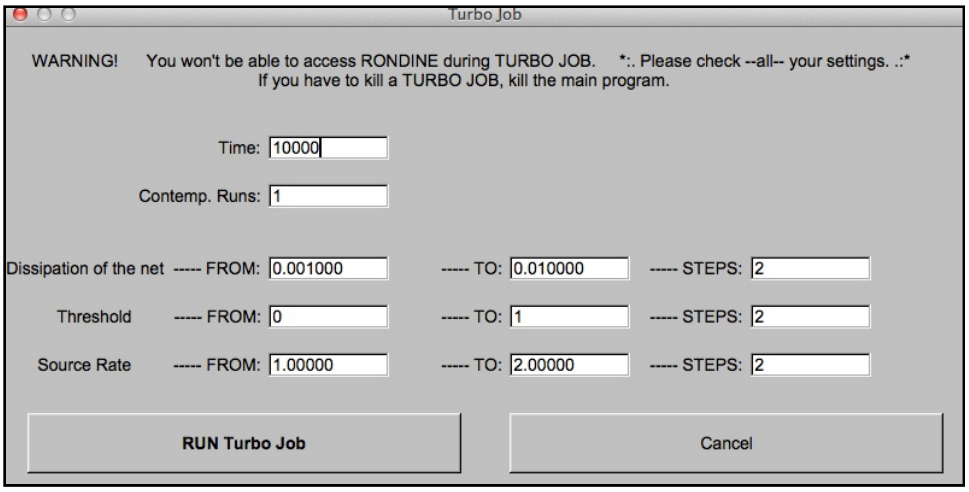
If you want to change the simulation's settings (thus starting a *new* simulation, which simulates a *new* system), you must push on the **CLEAR** button. This button resets the simulation to the given initial state and allow you to change the Dynamics' settings.

Here there is a fast explanation of the other functions you can find in the *Control Settings*' column:

- **Ticks** - *Displays the number of time-steps performed by the simulation.*
- **RUN** - *Runs the simulation*; it can be stopped by clicking again on it, or by clicking on the CLEAR button. It will in any case stop when the *ticks* reaches the value given in *maxtime.*
- **Run Step** - *Run a single step of the simulation.* This allow you to evolve the system for a small step of time, given in the input-text below. This is very useful if you want to see the evolution step-by-step, or if you want to check the state of your system after a while.
- **CLEAR** - *Cleans the simulation* and allows you to change the Dynamics' options. Note: this will not erase you output data: output is not over-written until you run a new simulation (after you cleared the last one).
- **DRAW** - *Yellow: draws the network; Grey: doesn't draw the network.* Since drawing the network is an operation that occupy computational resources, if not interested in the visualization you can just turn off the drawing option. The simulation will be fastest; if you need, you can reactivate the drawing option when you want.
- **Draw Options** - *Set what to draw, and what palette to use (only for nodes, fluxes palette is fixed.* In particular, you can decide to draw *fluxes.* Fluxes are displayed as triangular arrows and evaluate the total flux on the couple of links L_{ij} and L_{ji} (which are two different links); the direction of the arrow shows where the total flux is actually going (to i or to j), and the saturation indicates how actually intense is the flux.

- **TURBO RUN** - This is an important option. *TURBO RUN* starts a complete simulation for the system, from the given initial state, and runs it for *maxtime* steps. During this simulation the program is completely busy in the calculations: it does not display anything, nor it is possible to intetact with it. To the systems, it appears as a non responding program. However, doing so allows the program to compute the evolution of the system extremely faster than it does normally (up to 20 - 50 times faster). If the *print* option is checked, it will also prints out the data. Once the evolution is completed, ROnDINE will display the last state computed, and will be responding again.

- **T.** - This button opens a new dialogue window that allow you to perform a series of TURBO s by varying the given parameters (see Figure A.4). When the *dissipation* parameter varies, the network is re-built (see Sec. A.3.3). The details of the Turbo are printed on the console and on a file named "myout-AAA-T-txt". Each is printed as "myout-dAtBsC-K.txt", where A, b and C are integers counting the iteration done by changing the respective quantity (d: dissipation, t: threshold, s: source rate) and K is the output number (see Sec A.3.5).


Turbo Job

WARNING! You won't be able to access RONDINE during TURBO JOB. *:. Please check --all-- your settings. .:*
If you have to kill a TURBO JOB, kill the main program.

Time: 10000

Contemp. Runs: 1

Dissipation of the net ----- FROM: 0.001000 ----- TO: 0.010000 ----- STEPS: 2

Threshold ----- FROM: 0 ----- TO: 1 ----- STEPS: 2

Source Rate ----- FROM: 1.00000 ----- TO: 2.00000 ----- STEPS: 2

Figure A.4: ROnDINE's Turbo Dialogue

Display Options

If the LSS has been calculated, you can decide to display different states on the network: the *actual* state, the *stationary* (LSS) state, or the *difference* between the actual state and the LSS.

Load / Save State

You can load and save a state to/from a plain text file. The saved file is formatted as it follows: in the first line an integer N corresponding to the number of nodes of the network, in the second line an integer corresponding to the number of contemporary runs used, in the following N lines the number of particles for each node (average number if you used more than one run). The total number of particles is computed from the sum of the given data. To load a state, you must use a plain text file in the same format.

Data

In the first column of the right, you can see the "Data" section.

The big "**Data Window**" button shows and hide ROnDINE's data window.

Above this, you can see two check boxes: **Print Data** and **Print Corr.**, and both set and unset the possibility to print data on output files. Printing correlations, however, could result in very heavy files (since there are generalli N^2 correlations to print at each time step), and thus the possibility to not print them has been separated from the rest of the printable data.

Note that when running a *Turbo Run*, *Print Data* is automatically checked, while *Princ Corr.* is not: you can decide on your own if you desire to produce the correlation's data or not.

For further informations about how the data are printed, see Sec. A.3.5.

Network Options

This part of the column is devoted to control the network we are working on. As I said bedore, the network is separated from the dynamics, the network defines the *topology* there the particles are moving. There are several different option of networks that can be build using ROnDINE; it is also possible to *load* networks, previously saved using ROnDINE or artificially buildt.

- The white text rectangle display *the details of the network*, such as the type of network and the number of total nodes.
- **Layout** - throughout this button you can select different layouts to be used when displaying the network. The layout algorithms used are those provided by *igraph* library.
- **New Network** - this option allows you to build different kinds of networks. I will specify later what kind of networks ROnDINE can generate, and how the parameters required by the dials affects their structure.
- **Load** - Allows to load a Network from a particular formatted file (I named i *RONDINEfile*. Once the network is loaded, in the *path* dislayed above the main frame, you will be able to see where is stored the *RONDINEfile* storing the network you are working on.
- **Save - Save as** - Saves the network (and the actual state) on a *RONDINEfile*. The option *save* is available only if ROnDINE has a path to work on (obtainet through *load* or *save as* either.)
- **EXIT** - this is the button to *close ROnDINE*.

A.3.2 Data Window

In this window are collected all the on-run informations and graphs available.

On the top right, there is the **Draw** check-box. When checked, the graphs are drawn.

Density Histogram

This group allows you to have direct informations about the distribution of particles on the network, and to compute a *stop condition* for the simulation, based on the LSS. This means that this condition makes sense if and only if the system is *linear*.

The data box **requires LSS to be computed**. This is because the statistics computed rely on the fluctuations around the LSS.

Once you computed the LSS, you can see that there is an **histogram** displayed. This histogram is binned with the following values:

$$\rho_{norm}(i) = \frac{n(i)}{N\rho_{stat}(i)}$$

where $n(i)$ is the number of particles of the node i, N is the total number of particles, $\rho_{stat}(i)$ is the i-th component of the LSS, i.e. the component associated to the node i in the LSS.

If the network is in a stationary state, it is easy to show that the distribution of ρ_{norm} is a *Poisson distribution* with mean 1 and variance $\propto N^{-1/2}$.

Above the histogram displayed you can see the actual value of the mean and of the variance of the distribution.

Using the **STOP CONDITION** form, you can ask ROnDINE to stop when the histogram reaches a certain mean and a certain variance: ROnDINE won't, however, stop the first time it fulfill the stop condition, but after 10 times the condition is reached.

You can compute the most reasonable stop condition with the apposite button. In this case, ROnDINE will compute the average mean and average error on mean required to be sure to have reached the LSS in the *linear dynamics* case (i.e., if *threshold* $= 0$ *and constraint* $> N$.)

Activation

This graph shows the mean activation of the network at each time step. A single node i is **active** at the time t if it "*shooted*" particles to the other nodes.

If the mean activation of the network is 1 at time t, this means that *every node* in the net has passed at least one particle to a neighbour node in the time step from $t-1$ to t. If the activation is 0.5, only a half of the network's nodes is active.

The **Ticks to Display** box is used to determine how many time steps of the mean activation diagram will be displayed on the chart. The maximum possible is 10000.

Note: the computation of the activation as described in Sec. 4.5.1 is not yet fully implemented in the graphic interface. (See Sec. A.3.5 for an explanation)

Activation Histogram

Here the histogram of all the *mean activation values registered* is displayed.

Fluctuations Histogram

This histogram collects the fluctuations of the mean activations with respect to the *average activation*, which is computed by averaging every mean activation value registered from time $t = 0$ to the actual time.

A.3.3 Network constructors

The **New Network** button opens the dialogue window shown in Figure A.5.

Before we explore all the possibilities given by the dialogues, I must explain how is built a particular ensemble of networks, i.e. *networks with potential.*

Potential Networks

Some of the networks that ROnDINE can build have an *underlying potential.* In this kind of networks, to every node is assigned a *potential value.* This is similar to the potential you can have in a generic physical space.

Let us suppose we have node i with assigned potential V_i. Node i will have a certain number of neighbours, defined by the network constructor. But what is the weight of the links connecting i to its neighbour node labeled j (and thus, the transition rates π_{ij} and π_{ji})?

This is defined by the following relationship: *if node i and j are neighbours* (this must be defined previously), then

$$\pi_{ij} = e^{\frac{V_j - V_i}{2T}}$$

Where T is a "temperature" parameter that can be (and is) set to be $T = 1$.

A transition matrix built in this way will always satisfy the *detailed balance* property.

Dissipaton Option

For every possible type of network ROnDINE can build, you can always introduce a *dissipation*.
In every dialogue there is a **dissipation** check box, a *dissipation* input and a *Source Node* input.
This feature inserts a new node that serves as a reservoir (see Sec. 3.4).

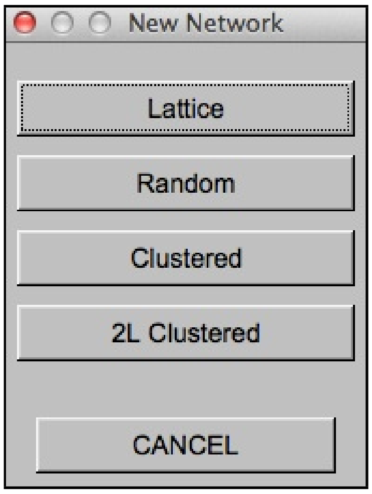


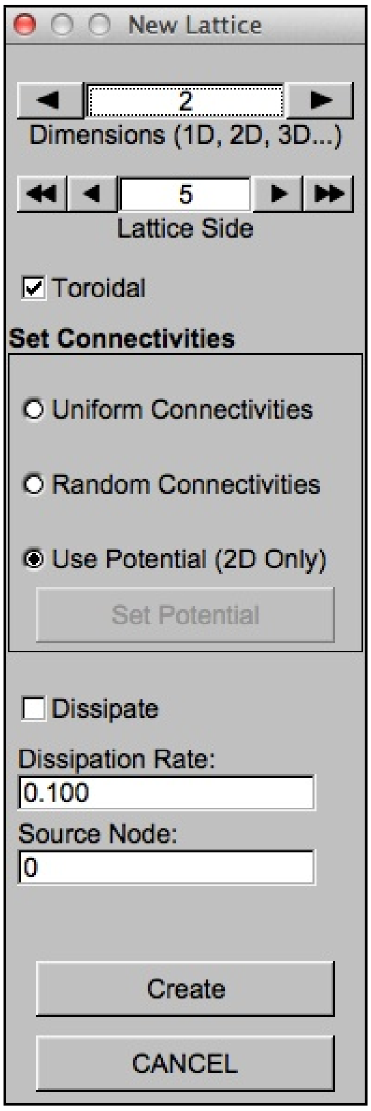
Figure A.5: New Network dialogue window

ROnDINE can generate **four** different kind of networks.

A.3.4 Lattices

ROnDINE can generate N-dimensional lattices with different properties. Figure A.6 shows the dialogue that generate lattices networks. As you can see, you can define the *Dimension* of the network (1D will be a line, 2D a square lattice, 3D a cube lattice..) and the *Side* of the lattice.
You can make id *toroidal* or not by checking the respective checkbox.
You can set three different kind of connectivities:

- **Uniform** - every link has the same weight
- **Random** - every link has a random value in $[0, 1]$, with uniform distribution on this interval



2
Dimensions (1D, 2D, 3D...)

5
Lattice Side

[✓] Toroidal

Set Connectivities

○ Uniform Connectivities

○ Random Connectivities

● Use Potential (2D Only)

☐ Dissipate

Dissipation Rate: 0.100

Source Node: 0

Figure A.6: New Lattice dialogue window

- **Use potential** - this is for 2D lattice only; it treats the 2D lattice as a discretization of a plane, defining in this way the *neighbours* of each node; then the potential used is a double-well potential on the plane.

The *set potential* button will allow you to use different kind of potentials. [Developing].

Random

Random networks are created as *Erdos-Reny random networks*, by setting the respective parameters. Tou can decide if the connectivities will be all equivalent or you can assign them a random (uniformly distributed in the $[0, 1]$ interval) value.

Clustered

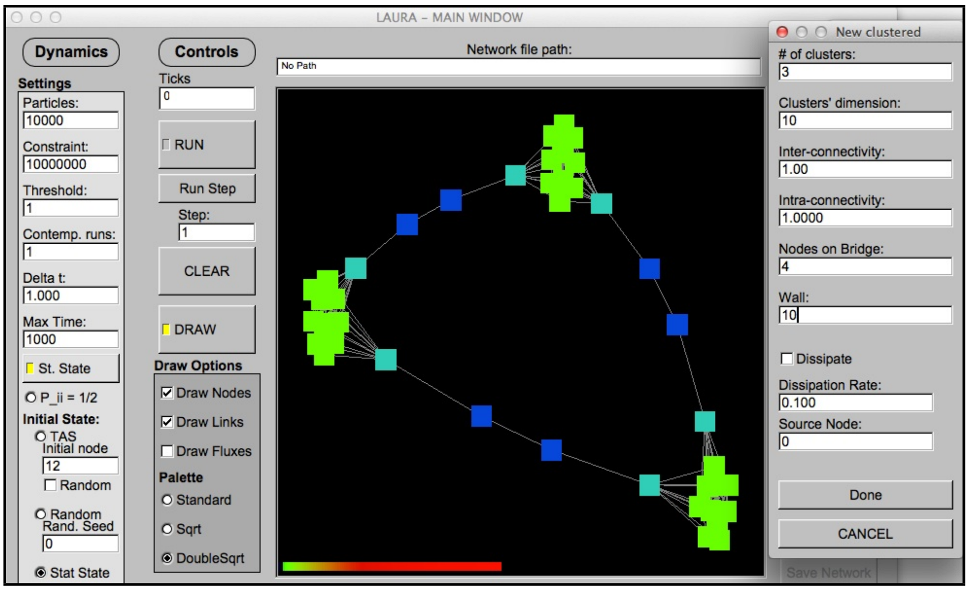
This is another kind of *potential network*.

A clustered network is a network made up of c clusters connected by $c(c-1)$ bridges. Each bridge is a linear chain of b nodes. Each cluster is a random Erdos-Reny newtork with N nodes, and connection probability p. You can set the values of N and p in the dialogue window, respectively *Cluster's Dimension* and *Inter-Connectivity*.

We have now M isolated clusters of N nodes. Clusters are hence connected among them by the *bridges*. As the names suggest, if a bridge connects the cluster k to the cluster l, not every node of the bridge is connected to the bridge, but only its extremes. One extreme of the bridge will be connected to the nodes in the cluster k, and the other extreme of the bridge will be connected do the nodes in the cluster l. The *Intra-Connectivity* parameter determine how many nodes of a cluster are connected to its afferent bridges. This is not a deterministic parameter, but similarly to the Erdos-Reny random networks constructor it determines a *probability to connect to the bridge*. For example, if we set the Intra-Connectivity to 0.5, every node in the clusters will have probability 0.5 to be connected (i.e. to be a neighbour) of the extremal node of any afferent bridge.

In this way, we have defined the neighbours of each node in the cluster. Thus, we set the potential: Cluster's nodes are then equipped with potential $V = 0$, while bridge's nodes are equipped with a potential given by the *Well* parameter in the constructor.

In Figure A.7 you can see an example of a Clustered Network in its stationary state:



Network file path: No Path

Settings
Particles: 10000
Constraint: 10000000
Threshold: 1
Contemp. runs: 1
Delta t: 1.000
Max Time: 1000
St. State
P_ii = 1/2
Initial State:
TAS
Initial node: 12
Random
Random
Rand. Seed: 0
[✓ Stat State]

Ticks: 0
Step: 1
[✓ Draw Nodes]
[✓ Draw Links]
Draw Fluxes
Standard
Sqrt
[✓ DoubleSqrt]

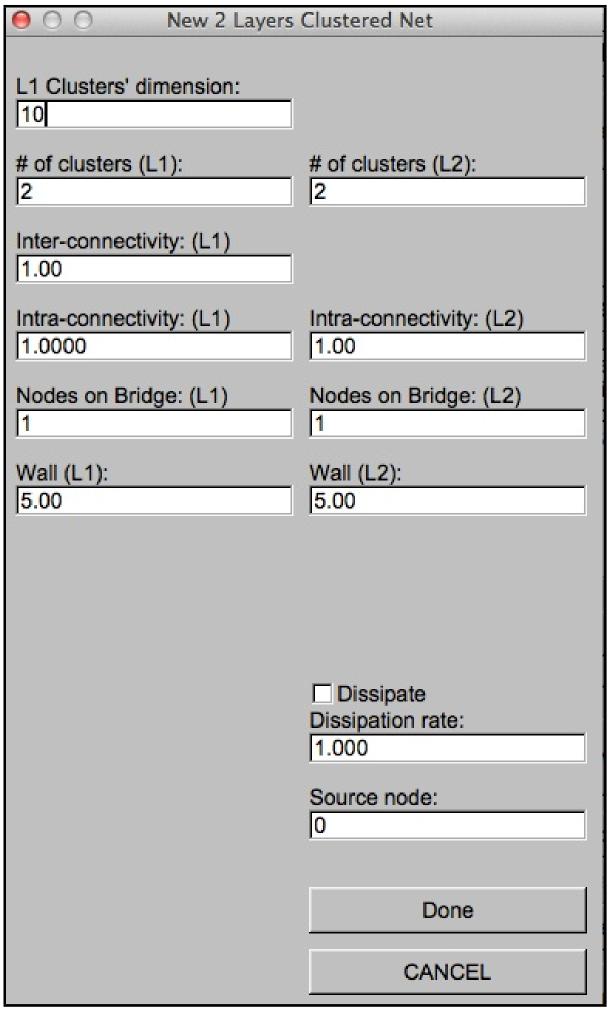
New clustered
# of clusters: 3
Clusters' dimension: 10
Inter-connectivity: 1.00
Intra-connectivity: 1.0000
Nodes on Bridge: 4
Wall: 10
Dissipate
Dissipation Rate: 0.100
Source Node: 0

Figure A.7: Example of Clustered Network; on the right are displayed the settings used to build it

2L Clustered

This constructor generate a two-layers Clustered network.

First of all, it generates a *Clustered Network*. This network is build according to the parameters of the left column in the dialogue window (Figure A.8). Then, this network is copied C_2 times, where C_2 is the number of the external clusters you desire. Each external cluster is then linked with the external


New 2 Layers Clustered Net

L1 Clusters' dimension: 10

# of clusters (L1): 2	# of clusters (L2): 2
Inter-connectivity: (L1) 1.00	
Intra-connectivity: (L1) 1.0000	Intra-connectivity: (L2) 1.00
Nodes on Bridge: (L1) 1	Nodes on Bridge: (L2) 1
Wall (L1): 5.00	Wall (L2): 5.00

☐ Dissipate

Dissipation rate: 1.000

Source node: 0

Figure A.8: Dialogue for 2L Clustered Network constructor.

bridges, as it was done with the Erdos-Reny clusters in normal Clustered Networks. The parameters for the external clusters and bridges are given in the right column.

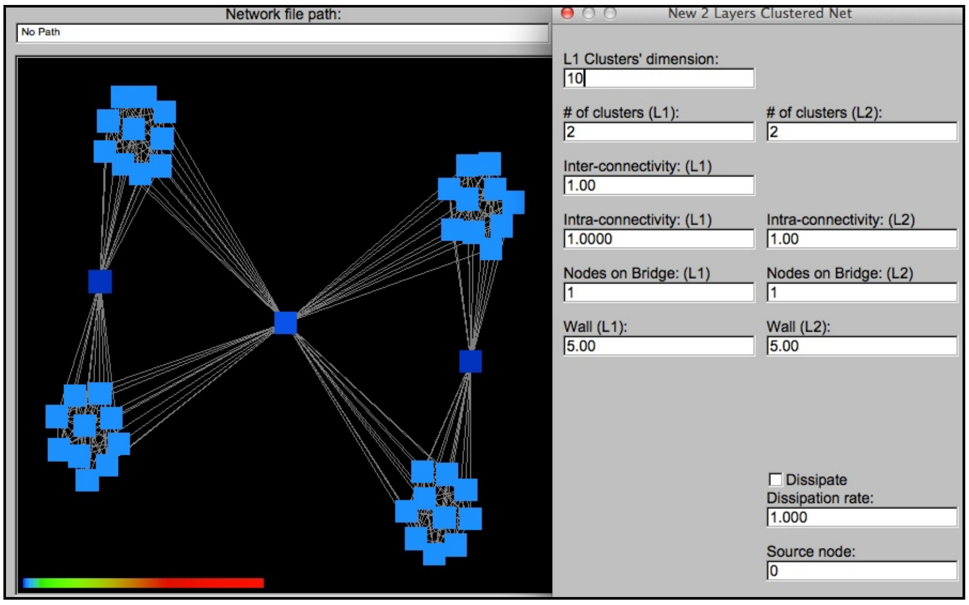
Figure A.9 shows an example of 2 Layers Clustered Network. As you can see, the inner bridges are not considered when connecting the external clusters throughout the external bridges.

A.3.5 Outputs

Whenever you run a simulation, using the **RUN** button or the **TURBO RUN** button, ROnDINE generates the output files and, if the *Print Data* checkbox is checked, prints on it.

ROnDINE will continue to write on the same files, until you use the CLEAR option. This options does not delete the output data, but tells ROnDINE to replace them with empty files when you run the new simulation.

Turbo-run automatically *clears* at its end.


Network file path:

No Path

New 2 Layers Clustered Net

L1 Clusters' dimension: 10

# of clusters (L1): 2

# of clusters (L2): 2

Inter-connectivity: (L1) 1.00

Intra-connectivity: (L1) 1.0000

Intra-connectivity: (L2) 1.00

Nodes on Bridge: (L1) 1

Nodes on Bridge: (L2) 1

Wall (L1): 5.00

Wall (L2): 5.00

☐ Dissipate

Dissipation rate: 1.000

Source node: 0

Figure A.9: Example of 2L Clustered Network, with respective constructor.

As I said previously, ROnDINE generates *seven* files which name will finish with i.txt where i is the index of the file.

For example, if I gave to ROnDINE the output name "myout", as it is setted by default, running a simulation will generate seven files: *myout1.txt*, *myout2.txt*, *myout3.txt*, *myout4.txt*, *myout5.txt*, *myout6.txt* and *myout7.txt*.

File 1

At each time step, ROnDINE prints:

$$t \quad \rho_1 \quad \rho_2 \quad \ldots \quad \rho_N$$

where t is the index of the time *tick*, N is the number of nodes (including the pump node, if present) and ρ_i is the density of particles on the node i at a time t, averaged on all the contemporary runs.

File 2

This file is written only if the LSS has been computed.

In case of a normal run, at each time ROnDINE prints

$$t \quad \Delta_1 \quad \Delta_2 \quad \ldots \quad \Delta_N \quad \Delta_{tot} \quad \hat{\Delta} \quad \delta$$

Where:

- Δ_i is the standard deviation (on all the contemporary runs) of $\rho_i(run)$ with respect to $\rho_{stat}(i)$
- $\Delta_{tot} = \frac{\sum_{i=1}^{N} \Delta_i}{N}$
- $\hat{\Delta}$ is the $L1$ distance between the actual state and the LSS

- δ is the $L1$ distance between the actual state and the previous

In case of a Turbo Run, the layout is almost the same. One more quantity is computed, Δ_{rel}:

$$t \quad \Delta_1 \quad \Delta_2 \quad ... \quad \Delta_N \quad \Delta_{tot} \quad \Delta_{rel} \quad \hat{\Delta} \quad \delta$$

Δ_{rel} is, as Δ_{tot}, a standard deviation, but it is computed using the normalized density,

$$\rho_{norm}(i) = \frac{n(i)}{N\rho_{stat}(i)}$$

.

File 3

This file is written only if we have *bridges*, i.e. we are working on a *Clusterd* (or *2L Clustered*) Network.

If we have n bridges, at each time step ROnDINE prints:

$$t \quad fl_1 \quad fl_2 \quad ... \quad fl_n$$

where ϕ_i is the total flux of particles (absolute value) on the i-th bridge.

File 4

This is the only files that is never deleted or overwritten; ROnDINE always appends on it. It is used to check on the statistical properties of the network.

ROnDINE prints on file 4 only when you push the **Print ^** button in the Data window, located under the density histogram, and will basically print informations about this histogram:

$$p \quad N \quad \bar{\rho}_{norm} \quad var(\rho_{norm}) \quad \frac{\bar{\rho}_{norm}}{N}$$

where:

- p is the number of particles in the network
- N is the total number of nodes in the network
- $\bar{\rho}_{norm}$ is the average value of the density histogram
- $var(\rho_{norm})$ is the variance of the density histogram
- $\frac{\bar{\rho}_{norm}}{N}$ should be constant for a fixed network, if the dynamics is linear.

File 5

This file collecys the intensity of the activation of each node, averaged on every contemporary run. At each time step ROnDINE prints:

$$t \quad s_1 \quad s_2 \quad ... \quad s_N$$

where s_i is the number of particles that note i has lost (and thus, "shooted") from time t_1 to t.

File 6

In this file the *correlations* are stored; this feature it not fully functioning, and it needs further work.

File 7

This file prints the mean activation $\bar{a}$ of the network. At every time step ROnDINE prints:

$$t \quad \bar{a}$$

The *mean activation* defined in 4.5.1 has been computed by modifying directly the source code; a proper integration with the interface will be developed in the near future

Bibliography

[1] G. Parisi *Complex Systems: a Physicist's Viewpoint* arXiv:cond-mat/0205297v1 [cond-mat.stat-mech]

[2] J Hesse, T. Gross, *Self-organized criticality as a fundamental property of neural systems.* Front Syst Neurosci, 2014 Sep 23;**8**:166. doi: 10.3389/fnsys.2014.00166. (2014)

[3] L. Arnold *Random dynamical systems*, Springer Monographs in Mathematics, Springer. (2003)

[4] Pierre Simon de Laplace, *A philosophical essay on probabilities* New York : J. Wiley; London : Chapman & Hall (1902)

[5] Kubo, *The fluctuation-dissipation theorem.* Reports on progress in physics **29**.1 :255 (1966)

[6] Rafael D. Vilela and Benjamin Lindner, *Comparative study of different integrate-and-fire neurons: Spontaneous activity, dynamical response, and stimulus-induced correlation.* Phys. Rev. E **80**, 031909 (2009)

[7] C.J. Perez, A. Corral, A.D. Guilera, K. Christensen, A. Arenas, *On the Self-Organized Criticality and Synchronization in Lattice Models of Coupled Dynamical Systems.* Int. J. Mod. Phys. **10**, Nos. 8 & 9, 1-41. (1996)

[8] Arbib, Michael A. *The handbook of brain theory and neural networks.* MIT press, 2003.

[9] C. Po Hsiang, John G Milton, Jack D. Cowan, *Connectivity and the dynamics of integrate-and-fire neural networks.* International Journal of Bifurcation and Chaos **4**.01 : 237-243. (1994)

[10] Choi, Hyung Wooc, Nam Jung, Jae Woo Lee. *Self-organized criticality of a simplified integrate-and-fire neural model on random and small-world network.* arXiv preprint arXiv:1405.4064 (2014).

[11] L. F. Abbott, *A network of oscillators.* Journal of Physics A: Mathematical and General **23**.16:3835. (1990)

[12] *igraph C library*, version 0.7 `http://igraph.org/c/`

[13] *FLTK C++ library*, version 1.3, `http://www.fltk.org/`

[14] Pierre Weiss. *L'hypothese du champ moleculaire et la propriete ferromagnetique.* J. Phys. Theor. Appl., **6** (1), pp.661-690. (1906) <10.1051/jphystap:019070060066100>

[15] J. R. Norris *Markov Chains* Cambridge Series in Statistical and Probabilistic Mathematics (No. 2), 1998, doi:10.2277/0521633966

[16] Michael William Newman,*The Laplacian spectrum of graphs.* Diss. University of Manitoba (2000)

[17] Gardiner, Crispin W. *Handbook of stochastic methods* Vol. 4. Berlin: Springer. (1985)

[18] Inomzhon Mirzaev, Jeremy Gunawardena, *Laplacian Dynamics on General Graphs.* Bulletin of Mathematical Biology V **75**, Issue 11, pp 2118-2149 (2013)

[19] see `http://www.netlib.org/lapack/explore-html/d9/d28/dgeev_8f.html` for LAPACK DGEEV references

[20] Nicolaas Godfried Van Kampen, **Stochastic processes in physics and chemistry**. Elsevier (1992)

[21] Rosenblatt, Frank *The Perceptron–a perceiving and recognizing automaton.* Report 85-460-1, Cornell Aeronautical Laboratory (1957)

[22] Pearson, *The Problem of the Random Walk.* Nature **72**, 294. (1905).

[23] P. Erdos, A. Rényi *On Random Graphs I.* Publicationes Mathematicae **6**: 290–297. (1959).

[24] R. Pawula, *Generalizations and extensions of the Fokker- Planck-Kolmogorov equations.* IEEE Transactions on Information Theory **13**: 33–41. (1967) doi:10.1109/TIT.1967.1053955.

[25] G. E. Uhlenbeck, L. S. Ornstein, *On the theory of Brownian Motion.* Phys. Rev. **36**: 823–841. (1930). doi:10.1103/PhysRev.36.823.

[26] Daniel T. Gillespie, *A rigorous derivation of the chemical master equation.* Physica A: Statistical Mechanics and its Applications **188.1** 404-425.(1992)

[27] T. Hida, *Brownian Motion*, Springer (1980)

[28] K. Itô *Stochastic integral*, Proc. Imp. Acad. Tokyo, **20**(8), 519–524.(1994)

[29] K. Itô *On a formula concerning stochastic differentials*, Nagoya Mathematical Journal, Vol **3**, 55-65 (1951)

[30] Michael E. Peskin, Daniel V. Schroeder *An Introduction to Quantum Field Theory.* Addison-Wesley (1995) ISBN 0201503972

[31] Hendrik Anthony Kramers, **Brownian motion in a field of force and the diffusion model of chemical reactions.** Physica **7**.4: 284-304. (1940)

[32] A. L. Hodgkin, A. F. Huxley, **A quantitative description of membrane current and its application to conduction and excitation in nerve**. The Journal of physiology 117 (**4**): 500–544. doi:10.1113/jphysiol.1952.sp004764. (1952).

[33] Albert, Réka, and Albert-László Barabási, *Statistical mechanics of complex networks.* Reviews of modern physics **74**.1: 47. (2002)

[34] L. F. Abbott, *Lapicque's introduction of the integrate-and-fire model neuron (1907).* Brain Research Bulletin, Vol. **50**, No. 5-6. (1999) doi:10.1016/s0361-9230(99)00161-6

[35] Nicolas Fourcaud and Nicolas Brunel, *Dynamics of the Firing Probability of Noisy Integrate-and-Fire Neurons.* Neural Computation archive Vol **14** Issue 9, pp 2057 - 2110 (2002)

[36] Naud, Richard *The Performance (and Limits) of Simple Neuron Models: Generalizations of the Leaky Integrate-and-Fire Model.* Computational Systems Neurobiology, Springer, pp 163-192 (2012)

[37] D.O *Hebb Distinctive features of learning in the higher animal*. In J. F. Delafresnaye (Ed.). Brain Mechanisms and Learning. London: Oxford University Press. (1961).

[38] Deepak Dhar, *Studying Self-Organized Criticality with Exactly Solved Models* arXiv:cond-mat/9909009v1 [cond-mat.stat-mech]

[39] Antal A. Jarai, *Sandpile Models.* arXiv:1401.0354v2 [math.PR]

[40] Bak P., Tang C. and Wiesenfeld K., *Self-organized criticality: An explanation of the 1/f noise.* Phys. Rev. Lett., **59**(1987) 381.

[41] Makoto Matsumoto, Takuji Nishimura, *Mersenne Twister (MT)* pseudo-random number generator, retrived at `http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/MT2002/emt19937ar.html`

Acknowledgements

My heartfelt thanks to my supervisor Prof. Armando Bazzani for his support during this work, and especially for his infinite patience.

My gratitude goes also to Prof. Giorgio Pastore from Physics Department of Trieste University, who has always a good advice to give; Prof. Feo V. Kusmartsev from Physics Department of Loughborough University, who gave me, some years ago, the boost I needed to continue my studies in Physics; and to Prof. Olivia Levrini from Physics Department of Bologna University, who, in her modules of Didactics and History of Physics, leaded us "through the Looking-Glass" of this wonderful and multifaceted subject.

Thank you to all my friends, to who's far away, to who's close, to who's both.

And ***Thank you Family****.*