Exploring the neural codes using parallel hardware

Javier Baladron Pezoa

UNIVERSITY OF NICE - SOPHIA ANTIPOLIS

DOCTORAL SCHOOL STIC

SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

P H D   T H E S I S

to obtain the title of

PhD of Science

of the University of Nice - Sophia Antipolis

Specialty : Computer Science

Defended by

Javier Baladron Pezoa

Exploring the neural codes using parallel hardware

Thesis Advisor: Olivier Faugeras

prepared at INRIA Sophia Antipolis, Neuromathcomp Team

defended on June 07, 2013

Jury :

Reviewers :	Markus Diesmann	-	Forschungszentrum Juelich
	Fred Hamker	-	Chemnitz University of Technology
Examinators :	Felix Schürmann	-	Blue Brain Project, EPFL
	Andrew Davison	-	UNIC, CNRS
	Pierre Kornprobst	-	Neuromathcomp Team, INRIA Sophia Antipolis

Abstract

The aim of this thesis is to understand the dynamics of large interconnected populations of neurons. The method we use to reach this objective is a mixture of mesoscopic modeling and high performance computing. The first allows us to reduce the complexity of the network and the second to perform large scale simulations.

In the first part of this thesis a new mean field approach for conductance based neurons is used to study numerically the effects of noise on extremely large ensembles of neurons. Also, the same approach is used to create a model of one hypercolumn from the primary visual cortex where the basic computational units are large populations of neurons instead of simple cells. All of these simulations are done by solving a set of partial differential equations that describe the evolution of the probability density function of the network.

In the second part of this thesis a numerical study of two neural field models of the primary visual cortex is presented. The main focus in both cases is to determine how edge selection and continuation can be computed in the primary visual cortex. The difference between the two models is in how they represent the orientation preference of neurons, in one this is a feature of the equations and the connectivity depends on it, while in the other there is an underlying map which defines an input function.

All the simulations are performed on a Graphic Processing Unit cluster. The thesis proposes a set of techniques to simulate the models fast enough on this kind of hardware. The speedup obtained is equivalent to that of a huge standard cluster.

Résumé

L'objectif de cette thèse est de comprendre la dynamique des grandes populations de neurones interconnectées. La méthode utilisée pour atteindre cet objectif est un mélange de modèles mésoscopiques et calculs de haute performance. Le premier permet de réduire la complexité du réseau neuronale et le second de réaliser des simulations à grandes échelles.

Dans la première partie de cette thèse une nouvelle approche du champ moyen est utilisée pour étudier numériquement les effets du bruit sur un groupe extrêmement grand de neurones. La même approche a été utilisée pour créer un modèle d' hyper-colonne du premier cortex visuel d'où l'unité basic, est des grandes populations de neurones au lieu d'une seule cellule. Les simulations sont réalisées en résolvant un système d'équation différentielle partielle qui décrit l'évolution de la fonction de densité de probabilité du réseau.

Dans la deuxième partie de cette thèse est présentée une étude numérique de deux modèles de champs neuronaux du premier cortex visuel. Le principal objectif est de déterminer comment les contours sont sélectionnés dans le cortex visuel. La différence entre les deux modèles est la manière de représenter des préférences d'orientations des neurones. Pour l'un des modèles, l'orientation est une caractéristique de l'équation et la connectivité dépend d'elle. Dans l'autre, il existe une carte d'orientation qui définit une fonction d'entrée. Toutes les simulations sont réalisées sur un cluster de processeurs graphiques.

Cette thèse propose des techniques pour simuler rapidement les modèles proposés sur ce type de machine. La vitesse atteinte est équivalente à un cluster standard très grand.

Acknowledgments

I would like to express my gratitude to my supervisor, Olivier Faugeras, for giving me the oportunity to do a PhD in his team. His vast knowledge and great support have been extremely important for the development of this thesis.

I would like to thanks the members of my commitee: Fred Hamker, Markus Diesmann, Andrew Davidson, Felix Schürmann and Pierre Kornprobst. All of you have spent part of your time in reading and commenting this document. I am sure that all the feed back that I will receive from you will improve this thesis.

I would also like to thanks my wife, Paulina Flores. Without her, I probably would not have ended this work.

This work was partially supported by the ERC grant #227747 NerVi, the FACETS- ITN Marie-Curie Initial Training Network #237955 and the IP project BrainScaleS #269921.

Introduction

The brain is an extremely complex system composed of a huge number of interconnected cells that together are able to efficiently solve hard problems like object recognition from images or motor control. Its amazing capabilities have attracted scientists from several different domains to start applying methods not common in biology to the study of neurons and realistic neural networks. Nowadays the use of approaches coming from mathematics, physics or computer science in neuroscience is usually called computational neuroscience.

This thesis belongs to the relatively new field of computational neuroscience. Our main objective was to study the dynamics of extremely large neural networks using mesoscopic models of brain activity and high performance computing. These new approaches come from areas far away from biology but we will show that they can be useful to improve our understanding of the brain at different scales.

Mesoscopic models and mean field approaches are used to reduce the complexity of the system. They allow us to take a detailed description of a neural network and reduce its number of equations to a quantity than can be treated numerically or analytically. This is specially useful for a complex system like the brain that is made of around 10^{11} neurons, each receiving signals from around 10^{4} other cells [Izhikevich 2007]. Just storing the complete amount of connections is almost impossible in current computers as it will require at least 10^{15} floating point numbers. Even very specific segments of the brain, like the visual cortex, are made of very large ensembles of neurons. This is why methods that reduce this complexity are required if we wish to understand how the brain works.

High performance computing allows us to make large simulations that would be impossible in current laptops or personal computers. The large computing power is used to extend numerical simulations to match real biological figures. Although with current technologies a simulation of the complete brain is extremely difficult, we can focus on one specific brain area, use mathematical techniques to reduce the complexity of the models and finally run a simulation to study its behavior. This simpler model of brain activity is usually still complex enough to require powerful hardware.

A large part of this work is focused on brain areas related to vision. One reason for this is that the primary visual cortex is one of the best studied brain area in biology. This is due to its position in the back of the head which makes experiments easier. Another reason for this is that the visual system is extremely efficient in solving a large number of task that are still very difficult for image processing or computer vision. This makes it an interesting system for reverse engineering as new ideas for algorithms may be based on its dynamics.

The first chapter of this thesis gives an overview of the three different domains that are involved in this work. It starts by introducing the reader to the biology of the brain by first describing the properties of single neurons and simple networks and then characterizing different areas of the brain cortex involved in visual processing. Then several mathematical models are presented that take into account all the information given previously. This second part also first deals with single neurons to then review mean fields and neural field methods. Finally, it shows current techniques in the field of high performance computing. Special emphasis is given to computing in Graphics Processing Units (GPUs) which is the main approach used in this thesis.

The second chapter of this thesis shows a set of numerical experiments that allow us to characterize the behavior of extremely large networks. This is done by first using a mean field reduction that transforms a system of stochastic differential equations, describing explicitly each neuron in the network, to a partial differential equation (PDE) which governs the evolution of the probability density of the complete group. This PDE is solved in a GPU cluster to characterize the dynamics of a single population. Then the simulations are extended to multipopulation models of the rat barrel cortex and the primary visual cortex of primates.

The third chapter of this thesis shows large scale simulations of two neural field models of the primary visual cortex. It first introduces the reader to a new model that doesn't use a feature based connectivity and shows the first numerical study of its behavior. Several predictions made on the original proposal and analysis of this model are rejected through numerical experiments. A second part of this sections, uses a spatial extension of a neural field model of one hypercolumn (group of neurons that represent the different possible orientations in a region of an image) to study the effects of long range connectivity in edge enhancement and perceptual grouping. An important result presented in this section are the techniques used for a fast simulation in GPUs of the corresponding neural field models.

A digital version of this document together with several movies that complement the results presented in Chapter 2 can be found in: http://www-sop.inria.fr/members/Javier.Baladron/thesis.html

Introduction (version française)

Le cerveau est un système extrêmement complexe composé d'un grand nombre de cellules interconnectées qui ensemble sont capables de résoudre efficacement de difficiles problèmes comme la reconnaissance d'objet en images. Ses surprenantes capacités ont attirés des scientifiques de différents domaines qui ont commencé à appliquer des méthodes peu habituelles en biologie pour l'étude de neurones et des réseaux neuronales réalistes. Aujourd'hui l'utilisation d'approches qui viennent des mathématiques, de la physique, de l'informatique ou autres domaines similaires en neuroscience s'appellent neurosciences computationnelles.

Cette thèse appartient à la relative nouvelle aire de neurosciences computationnelles. Notre principal objectif fut d'étudier des réseaux neuronaux extrêmement grands en utilisant des modèles mésoscopiques d'activité cérébrale et de calculs de haute performance. Ces nouvelles approches viennent de domaines éloignés à la biologie mais nous démontrerons qu'elles peuvent être utiles pour améliorer la compréhension du cerveau à différentes échelles.

Des modèles mésoscopiques et de champs moyens sont utilisés pour réduire la complexité du système. Ceci va permette de réduire le nombre d'équations d'une description détaillée d'un réseau neuronal à une quantité qui peut être traité analytiquement ou numériquement. C'est spécialement utile pour un système complexe comme le cerveau qui est composé d'environ 10^{11} neurones, chacune recevant des signaux d'environ 10^4 autres cellules [Izhikevich 2007]. Seulement enregistrer la totalité des connections dans les ordinateurs actuelles est presque impossible car on nécessiterait enregistrer 10^{15} virgules flottantes. D'ailleurs des segments très spécifiques du cerveau comme le cortex visuel sont composées de groupes très grands de neurones. C'est pour cette raison que nous avons besoin de méthodes pour réduire la complexité pour comprendre comment fonctionne le cerveau.

Le calcul de haute performance nous permet de réaliser de grandes simulations qui sont impossibles à réaliser sur un ordinateur personnel. La puissance de calcul de ce type d'ordinateur est utilisée pour étendre les simulations numériques et atteindre des quantités réalistes. Puisque qu'avec les technologies actuelles une simulation du cerveau entier est très improbable, il fait se diriger vers une zone spécifique, ensuite utiliser des techniques pour réduire la complexité. Malgré tout ce modèle simplifié, reste suffisamment compliqué pour nécessiter un hardware puissant.

Une grande partie de ce travail est dirigée vers l'aire du cerveau destiné à la vision. Une des raisons est que le cortex visuel est une des zones les plus étudiées du cerveau en biologie. Cela est dû au fait qu'elle se trouve à l'arrière de la tête et rend les expériences plus faciles. Une autre raison pour cela est que le système visuel est extrêmement efficient pour résoudre un grand nombre de problèmes qui sont encore plus difficile à résoudre pour le traitement d'image numérique ou la vision par ordinateur. Ceci le rend intéressent pour l'ingénierie inverse car de nouveaux algorithmes peuvent être basés sur son comportement.

Le premier chapitre de cette thèse présente une vue d'ensemble des trois différents domaines impliqués dans ce travail. Il commence par introduire le lecteur à la biologie du cerveau décrivant d'abord les propriétés d'un neurone et d'un réseau simple. Puis caractérise les différentes aires du cortex impliquées dans le processus visuel. Ensuite des modèles mathématiques sont présentés et prennent en comptent l'information donnée précédemment. Cette seconde partie commence également par des neurones uniques et ensuite passe par des modèles de champs moyens et de champs neuronaux. Finalement montre des techniques actuelles dans le champ du Calcul de haute performance. Une importance spéciale est donnée aux calculs sur des processeurs graphiques (GPU) qui est la principale technologie utilisée dans cette thèse.

Le deuxième chapitre de cette thèse montre une série d'expériences numériques qui a permis de caractériser le comportement de grands réseaux neuronaux. Ceci est réalisé d'abord en utilisant une réduction de champs moyen qui transforme un système d'équation différentielle stochastique, qui décrie explicitement chaque neurone dans le réseau, en une équation différentielle partielle qui gouverne l'évolution de la densité de probabilité de l'ensemble du groupe.

Le troisième chapitre de cette thèse montre des simulations à grandes échelles de deux modèles de champs neuronaux du premier cortex visuel. Premièrement il introduit un nouveau modèle de champs neuronaux qui n'utilise pas une connectivité basé sur les caractéristiques d'équations et montrent une première étude de son comportement. Plusieurs prédictions faites dans la proposition initiale du modèle ont été rejetées par des expériences numériques. La deuxième partie de ce chapitre utilise une extension spatiale d'un modèle de champs neuronaux d'une hyper-colonne (groupe de neurones qui représente les différentes orientations possibles dans une section de l'image) pour étudier les effets de la connexion à longue distance. Un résultat important présenté dans ce chapitre, sont les techniques utilisées pour une simulation rapide en GPU.

Contents

Chapter 1

Biology and Computation

Contents

The aim of this chapter is to provide an overview of the three different but related topics that are the core of this thesis. We start by describing some elements of the nervous system, focusing on its visual processing areas as these are the main objects of study in the computational experiments that are presented in the following chapters. Some modeling techniques are presented in the following section. These will allow us to transform part of the biological facts into equations, making possible an analytic or numerical study of the visual system. Finally we describe the computational techniques that will allow us to solve these equations. Simulating these kind of models requires very powerful computers and modern parallelization techniques which are the focus of thes last section.

The goal of this chapter is not to provide a complete detailed introduction to the biology of vision, nor of its mathematical modeling. Here, only the theory and facts necessary for understanding the results of this thesis are presented. For a complete review on the biology of neural science we recommend the book [Kandel 2000] and for a more theoretical perspective [Dayan 2001, Gerstner 2002, Ermentrout 2010].

1.1 Overview of the visual system

In this section we describe the basic structure of the first areas of the nervous system in charge of processing vision. We start by describing the most basic unit of the brain, a neuron, and then continue to more complex networks that can be found in the cerebral cortex.

1.1.1 Neurons

Neurons are the basic component of the brain and the nervous system. It is through a limited number of this kind of cells (around 10^{11}, [Izhikevich 2007]) that the brain develops all of its amazing capabilities. Although a great variety of neurons have been found in different parts of the brain they all have a set of common characteristics that will be described next.

What makes neurons special is their ability to receive and send electrical signals from and to long distances. In order to achieve this, each cell is composed of dendrites, which form a structure to receive the input from other cells, an axon, which is a tree-like structure that sends signals to other neurons, and a soma, which is the nucleus of the cell. The main difference between neuron types is in the shape of its dendrites [Ermentrout 2010]. Figure 1.1 shows a representation of a set of neurons and how axons on one cell contact the dendrites in another.

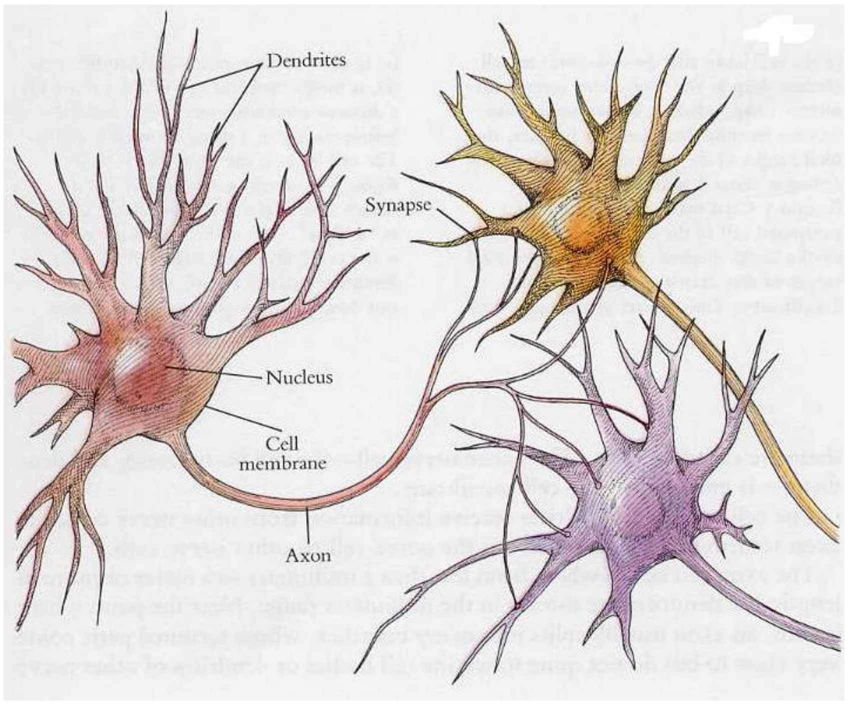


Figure 1.1: Neuron structure and interconectivity. From [Hubel 1995]

The soma is in charge of integrating the signals received from other neurons through connections called synapses. Each signal received changes the membrane potential of the neuron (whic is the difference in voltage between the interior and exterior of the cell) and if the change is big enough a spike is produced. If the change makes the membrane potential more positive (or less negative) the cells has depolarized while if it has make the opposite change the cell has hyperpolarized. The spike (also called action potential), or abrupt change, is transported through the axon to other cells.

Synapses can be of 2 types: electrical or chemical. This first is rare and is a direct and very fast connection between the 2 cells. In electrical synapses there are special channels, called gap junctions, that are capable of transporting current and to induce directly a voltage change in the postsynaptic neuron.

Chemical synapses are more common and when a spike from the emitting cell (presynaptic neuron) arrives, neurotransmiters are released into a small space that exists between the axon and the dendrites of the receiving cell (postsynaptic neuron), called the synaptic cleft. This transmitter is bound to receptors in the dendrites that finally produce the change in membrane potential. The effect produced by this kind of synapse depends on the type of neurotransmiter released and it may excite or inhibit the cell in very complex forms.

1.1.2 Retina

The retina is the first part of the central nervous system in charge of processing the visual information that enters the eye. It is located in the back of the eye so it receives the light as soon as it has passed through the lens. Only a few types of neurons are present in it: photoreceptors, horizontal cells, bipolar cells, amacrine cells and ganglion cells.

The photoreceptor cells are neurons that activate themselves in the presence of light. There are mainly 2 types: rods and cones. Rods are cells specialized in dim-light vision, while cones are specialized for situations where a larger amount of light is present. During the day it is mostly the cones that are active, but at night our vision is mediated mainly by rods.

The retina is formed first by a layer of photoreceptor cells, followed by a synaptic layer that connects this first group of neurons to a layer of bipolar and horizontal cells. This second layer is then connected through another synaptic layer to ganglion cells, which send the final output of the retina to the rest of the brain. Bipolar cells can also connect to amacrine cells, which can connect to ganglion cells on the next layer or send information back to other bipolar cells. This layered structure is the main organization of the retina and can be seen in figure 1.2.

Each bipolar cell has a receptive field, which is the zone of the visual field (the total area in which objects can be seen) for which a stimulus provokes a change in its membrane potential. Some of these cells depolarize when a small spot stimulus is presented in the center of their receptive field (ON-center type) and others are hyperpolarized by the same stimulus (OFF-center cells). Other types of bipolar cells can also be found in several species. Nonetheless all of these are subtypes of ON or OFF [Nelson 2004]. The different subtypes come from the consideration of other features such as connectivity with photoreceptors.

Ganglion cells also present receptive fields but their response to a stimulus is more complex. They are the only cells in the retina that generate action potentials. Typically they present a center-surround configuration and are normally divided into ON-cells, OFF-cells and ON-OFF cells. The ON-cells activate when a spot of light is present in the center of the receptive field and keep spiking during the whole duration of the stimulus. The OFF-cells do not generate spikes during a stimulus in the center of their receptive fields but produce sustained activity when they are turned off. The ON-OFF cells produce small burst of spikes when the stimulus is turned on or off. Other types, that are selective to different characteristics of the input can also be found, but the ON-cells, OFF-cells and the ON-OFF cells are the most prominent [Cleland 1974].

For more details on the retina, see [Masland 2001a, Masland 2001b, Wassle 2004, Wohrer 2008].

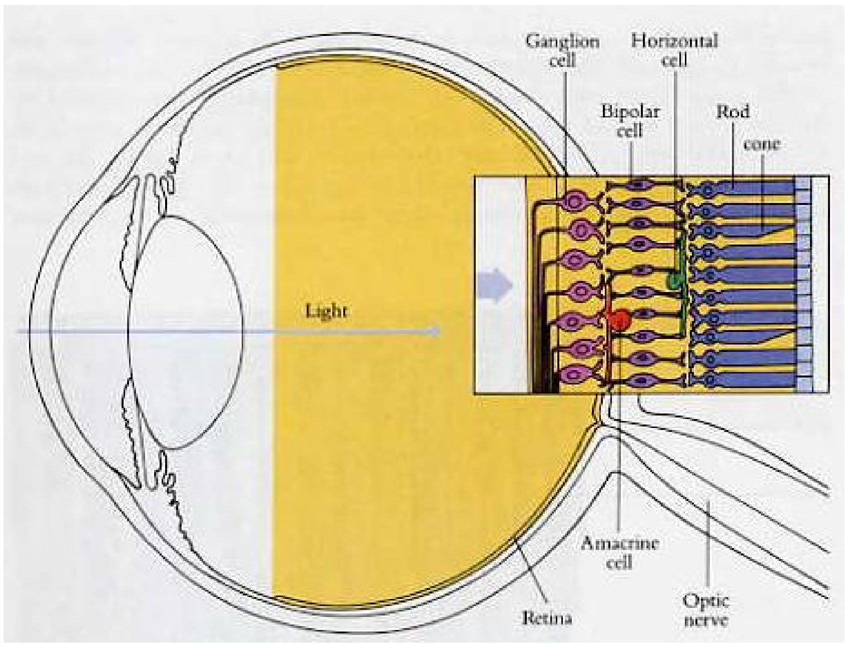


Figure 1.2: Position of the retina and the back of the eye and its layered structure. From [Hubel 1995].

1.1.3 Lateral geniculate nucleus

The ganglion cells in the retina connect with the Lateral Geniculate Nucleus (LGN) in the Thalamus. It serves as the main relay of information coming from the retina to the cortical areas of the brain, where more complex analyses of the visual input are made.

The LGN is formed by 6 layers, the 4 upper layers are made of smaller cells and are called the parvocellular layers while the other 2 are made of larger cells and are called the magnocellular layers [Kaplan 2004]. Morphologically different ganglion cells connect to different layers, generating different paths of information.

The P-path starts at the retinal midget ganglion cells, which receive input from bipolar cells which connect to only one single cone photoreceptor. These ganglion cells connect to the parvocellular layers of the LGN. The M-path starts at the parasol ganglion cells of the retina, which receive input from several bipolar cells which are connected to several photoreceptors. This second type of ganglion cell connect to the magnocellular layers of the LGN. There is a third, less known, path, called the K-path, that connects the retina with small layers that are intercalated between the main parvocellular and magnocellular layers of the LGN. This third smaller type of layer is called koniocellular. For a detailed comparison between the 3 kind of layers and their connectivity see [Xu 2001]. A diagram of these connections is presented in figure 1.3.

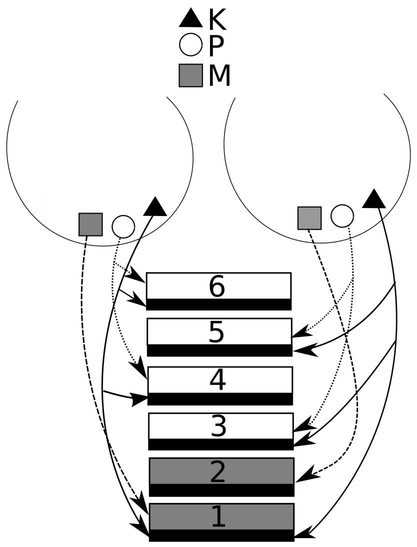


Figure 1.3: Connection from ganglion cells to the different layers in the LGN. The diagram shows how each different type of ganglion cell connects to different layers. Also each layer receives input from only one eye. Redrawn and adapted from [Sherman 2004]

For a detailed review on the structure of the LGN see [Sherman 1996].

1.1.4 Primary visual cortex

All the pathways in the LGN connect to the primary visual cortex or V1, the largest of a group of cortical areas devoted to visual processing. This is one of the best known areas of the brain due to its size and to its position at the back of the head which makes biological experiment easier than in other visual regions of the brain.

As the LGN and other areas of the cortex, V1 has a layered structure composed of 6 layers. Most of the input from the LGN is received in layer 4, where the different paths connect to different regions. More details on the connectivity between the LGN and V1 (in the macaque monkey) can be found in [Callaway 2004].

1.1.4.1 Receptive fields

Probably the most interesting thing about the primary visual cortex is the presence of receptive fields of a greater complexity than their predecessor neurons in the LGN. This was originally discovered by Hubel and Wiesel and presented in a seminal paper [Hubel 1962]. They detected neurons with 2 different types of receptive fields: they called them simple cells and complex cells. Simple cells have elongated receptive fields that can be separated into 2 regions, one that excites the neuron when light is presented in it and another region that inhibits the cell. This behavior is similar to a linear filter, where the output is the sum of the negative and positive areas. Complex cell receptive fields can't be separated into different regions and they are believed to realize non-linear operations over the input image. The behavior of complex cells can be seen as a non-linear combination of a set of linear filters applied to the input image [Rust 2005].

Two nearby neurons in V1 have receptive fields that represent nearby sections of the visual field. In this way, the cells in the primary visual cortex create a complete map of the visual field, called the retinotopic map.

As can be seen in the examples given in figure 1.4 the receptive fields of simple cells are elongated and tilted, which are very different from the circular receptive fields of LGN cells and retinal ganglion cells. This inclination is not the same in all cells and produces the existence of a preferred orientation, i.e. the angle of a bar on the input image that produces the maximum activity. LGN neurons do not have this property, so this must arise from the connectivity of cells between the thalamus and the primary visual cortex. With current neuroimaging methods a map of preferred orientation can be obtained and its structure can be studied [Okamoto 2011, Ts'o 1990, Slovin 2002]. An example of the kind of map than can be produced with these modern techniques can be seen in figure 1.5.

Maps show a continuous structure of orientations except at points where all the different preferences converge, called pinwheels. The study done in [Ohki 2006] shows that these kind of points are singularities in the map and their positions generate an organized structure. It also shows that neurons close to a pinwheel center have a sharper tuning curve.

1.1.4.2 Neuron selectivity

Neurons in the primary visual cortex do not only present a preferred orientation but they also favor other, specific, attributes of the visual field. These preferences are not present in LGN cells, so the synapses between them and V1 are the main tools

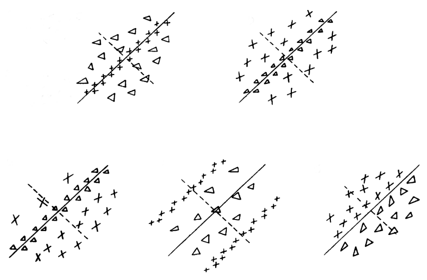


Figure 1.4: Receptive field from simple cells in V1. The x's show areas of excitatory responses and triangles show areas of inhibitory responses. Adapted from [Hubel 1962]

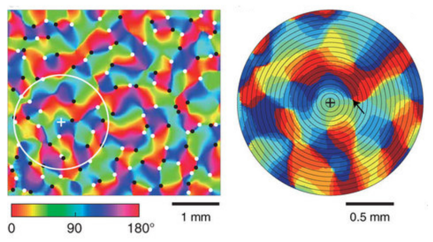


Figure 1.5: Left: orientation map found in the monkey V1. Right: zoom of the same map. Taken from [Okamoto 2011]

for computing these attributes. Current brain imaging techniques can normally produce maps for each preferred attribute in a similar as is done for orientations. Some of the characteristics that have been found to dominate the activation of V1 neurons will be described next.

Cells in V1 normally receive inputs from both eyes, differing from LGN neurons where each layer is associated with a particular eye (see figure 1.3). It is in this piece of the cortex that the information from the two eyes converges and the difference between both is detected. Neurons with ocular preference were already detected by Hubel and Wiesel as described in [Hubel 1977]. The data they obtained show that most of the cells are mainly driven by one eye, and that the amount of neurons that respond equally to both eyes is small. Several examples of ocular dominance maps obtained with modern optical imaging can be seen in figure 1.6.

This disparity preference in the output of simple cells can be understood as the sum of the application of two linear filters, one for each eye. Although at the time when Hubel and Wiesel discovered this selectivity it was believed that the 2 filters were the same, current studies show they may differ and that this difference may explain the binocular activity of cells ([Anzai 1999]). A review of the history of this discovery and more details on the disparity preference in the visual system can be found in [Cumming 2001].

Explaining ocular preference in a similar way for complex cells is much more complicated and several models have been proposed [Cumming 2001]. A model that provides an explanation for most of the experimental results was proposed in [Ohzawa 1990]. In this model complex cells are seen as the combination of a set of simple cells with different disparity tuning.

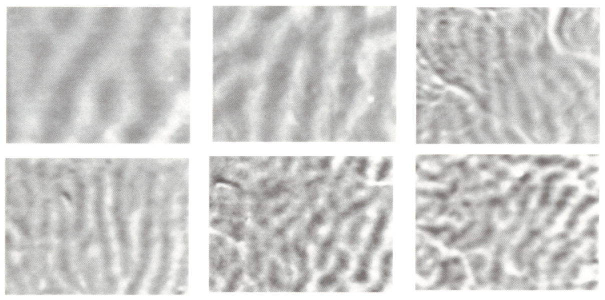


Figure 1.6: Ocular preference maps from six different monkey subjects, taken from [Obermayer 1993]. Dark areas indicate a stronger response to stimulation of the right eye, bright areas indicate stronger response for stimulation to the left eye, and gray areas indicate equal response to stimulation of either eye.

Neurons in the primary visual cortex are also selective for the direction of motion. Three types of selectivity can be found in V1: non directional cells, directionally biased neurons and motion opponent cells [Clifford 2003]. Non directional cells only feature a preferred orientation but not a preferred direction. In a directionally biased neuron a bar of the preferred orientation passing through the receptive field following a direction perpendicular to this orientation will cause the highest level of activity. A motion opponent cell respond strongly to its preferred direction but its inhibited by motion in the opposite direction. An example of a direction map obtained by optical imaging can be seen in figure 1.7

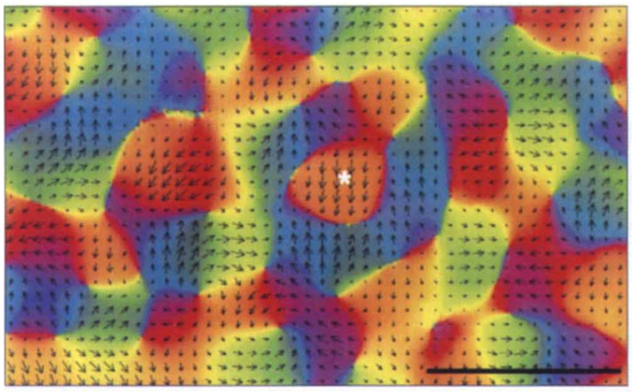


Figure 1.7: Direction selectivity map in a monkey subject, taken from [Weliky 1996]. The direction of best response is color coded and represented by arrows whose length indicates the size of the response.

Optical imaging also show that neurons present preferences for a spatial frequency presented on its receptive field. The experiments presented in [Issa 2000] show high resolution maps of preferred spatial frequencies at the preferred orientation. This map show a wide range of different preferences and a continuous change with the presence of pinwheels, similar to orientation maps.

1.1.4.3 Columnar organization

Aside from the horizontal layered structure, the primary visual cortex has a vertical organization. A similar response is found for neurons located in any line perpendicular to the surface of the cortex. The properties of the neurons, like orientation or eye preference, change when moving across the surface of V1 but do not change much with depth [Hubel 1977]. This structure is known as a cortical column.

Hubel and Wiesel also proposed another structure called the hypercolumn. The hypercolumn is made of a group of columns that contains all orientations for both eyes. This concept has been the basis of several mathematical models of V1 [Hansel 1997, Bressloff 2001b].

Columns are connected to other nearby columns by synapses whose strength depends on the distance between the 2 connecting elements [Das 1999]. This short range interaction connects each element to its neighbors independent of their preferred orientation. Like this, the position in the orientation map of the cell is critical in the computation it makes. Each cell is inhibited by cells in its neighborhood that have a perpendicular orientation preference and excited by similar orientation preferences.

Neurons in V1 also send long-range lateral connections to other columns that contact excitatory and inhibitory cells forming patches of terminals [Lund 2003]. This lateral connectivity is anisotropic, i.e. it follows the direction of the preferred orientation and connects only with similar elements [Angelucci 2002].

More detail on intra and inter column connectivity can be found in [Tucker 2004].

1.1.4.4 V1 and beyond

The complexity of the visual system is huge as there are 32 neocortical areas involved in vision processing, where 7 of them are also involved in other tasks [Felleman 1991]. The primary visual cortex is believed to be the first one of these, and to be in charge of extracting information content from the images that is necessary for other higher cognitive functions that are related to other higher visual areas of the cortex. Like this, the computation done in V1 extract features that serve as input for more complex functions.

An example of the above are directional selective neurons in V1 which connect to area MT, a piece of the cortex believed to be highly specialized in motion processing [Clifford 2003]. Almost all of the cells in MT present a preferred direction of motion and its output signal is associated to eye movements and optic flow processing. For more details on the area MT see [Born 2005].

Cells in the primary visual cortex also connect to visual area V2. In this projection the different paths are kept segregated, i.e. areas of V1 whose input is dominated by a similar path connect to a specific layers of V2 [Sincich 2002]. It has been shown that V2 processes more complex geometrical forms than V1. Neurons in V2 may present preferences to shapes like circles, crosses or others figures composed of several edges or lines [Hegde 2000].

Neurons in V1 also connect to area V3. This is a less known area due to its deeper position in the cortex which makes biological experiment hard. This area has been divided in several regions that receive connections from different cells in V1 [Felleman 1997]. The ventral half carries a representation of the upper visual field and receive projections from neurons that are selective for orientation and wavelength but not for direction while the dorsal half carries a representation of the lower visual field and receive input from cells that are selective to direction of motion. One of the main conclusion of the study of the functional properties of V3 presented in [Gegenfurtner 1997] is that its function must be to integrate information.

Areas V2 and V3 connect also to area V4 [Ungerleider 2008]. An interesting feature of this area is that the receptive field properties of its neurons are strongly affected by attention signals ([Connor 1997, McAdams 1999]). Theoretical studies have shown that a feedback (or attention) signal coming from cells associated with eye movement may tune the receptive field structure in V4 and improve its object recognition capabilities ([Hamker 2006, Hamker 2008, Zirnsak 2010]). In this case, the number of receptive field increases around the planned saccade (fast eye movements).

1.2 Mathematical models of brain activity

In this section we describe several theoretical approaches that transform the biological facts described previously into mathematical models of brain activity. These models allow a better understanding of the phenomena occurring in the brain through the use of several different approaches that are outside the boundaries of classical biology. As measurements of real neurons are difficult to obtain, this kind of technique is becoming a standard tool in neuroscience.

The first part of this section deals with mathematical descriptions of a single neuron before introducing techniques designed to deal with the complexity of large scale networks. In order to understand correctly the brain we need not only to comprehend the dynamics of its forming units but also how they interact with each other.

1.2.1 Single neuron models

Here we describe several approaches to modeling the dynamics of just one single neuron. We start with the realistic but complex Hodgkin-Huxley model and then present some simplifications of these equations. These other simpler models are widely used in the computational neuroscience community as they can explain lots of biological phenomena with a lower level of complexity.

1.2.1.1 Hodgkin-Huxley model

Alan Lloyd Hodgkin and Andrew Huxley [Hodgkin 1952] proposed in 1952 a mathematical model to explain the generation of action potentials in the cells of the giant squid. This model considers the neurons membrane potential, i.e. the voltage difference between the outside and the inside of the cell, which varies through time. The voltage changes due to the existence of permeable ion channels, through which positive or negative ions enter or leave the cell. When a positive ion enters the neuron the membrane potential rises and when it is a negative ion it is reduced. The Hodgkin-Huxley model considers 3 kinds of ions that a cell exchanges with its exterior: Na^+, K^+ and Cl^-.

If no external current is applied to the cell, the electrical potential and concentration difference induces the flow of ions until an equilibrium is reached. The value of this equilibrium potential is different for each ion and is given by the following Nernst equation:

$$\mathrm{E}_{ion} = \frac{RT}{zF} \ln \frac{[\mathrm{Ion}]_{out}}{[\mathrm{Ion}]_{in}}, \tag{1.1}$$

where: $[\mathrm{Ion}]_{out}$ is the concentration of the ion outside the cell, $[\mathrm{Ion}]_{in}$ is the concentration of the ion inside the cell, R is the universal gas constant (8,315 mJ/(K°

Mol)), T is the temperature in degrees Kelvin, F is the Faraday constant (96,480 Coulombs/Mol) and z is the valence of the ion.

Using the previous facts, a neuron can be represented as an equivalent electrical circuit where each channel is seen as a battery connected to a resistor and the membrane as a capacitor. A diagram of this distribution can be seen in figure 1.8. Using Kirchhoff's law the current flowing through the membrane potential can be computed as the sum of all the individual currents:

$$I = C\dot{V} + I_{Na} + I_K + I_{Cl}. \tag{1.2}$$

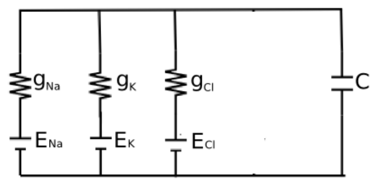


Figure 1.8: Circuit representing one neuron. Each ion potential is represented by a battery connected to an ion channel, or resistor. The membrane is represented as a capacitor.

The conductance of each channel depends on the time and on the current voltage, except for the Cl^- or leakeage channel. Each of them has a maximum conductance that we denote by: g_{Na}, g_K and g_{Cl}. 3 other variables are defined, m, n and h, that control the probability that each type of channel is open, giving finally the Hodgkin-Huxley equation for the membrane potential:

$$C\dot{V} = -g_{Na}m^3h(V - E_{Na}) - g_K n^4(V - E_K) - g_{Cl}(V - E_{Cl}) + I_{ext}. \tag{1.3}$$

The variables m, n and h vary according to the following equations, also defined by Hodgkin-Huxley. The α and β functions are defined in table 1.1.

$$\begin{aligned} \dot{m} &= \alpha_m(V)(1 - m) - \beta_m(V)m \\ \dot{n} &= \alpha_n(V)(1 - n) - \beta_n(V)n \\ \dot{h} &= \alpha_h(V)(1 - h) - \beta_h(V)h. \end{aligned} \tag{1.4}$$

Figure 1.9 shows the solution of the equation when a constant external current is applied to the neuron after 1000 milliseconds. Before the input the neuron stays at a constant potential, its resting state. As soon as the input starts the cells begins to emit spikes. After each spike the neuron stays for a moment at a very low potential before starting to increase slowly (depolarization) until an action potential is produced.

For more details on this model see [Gerstner 2002, Ermentrout 2010, Izhikevich 2007, Dayan 2001]

x	$\boldsymbol{\alpha_x(u/mV)}$	$\boldsymbol{\beta_x(u/mV)}$
m	$(0.1-0.01u)/(e^{1-0.1u}-1)$	$0.125e^{-u/80}$
n	$(2.5-0.1u)/(e^{2.5-0.1u}-1)$	$4e^{-u/18}$
h	$0.07e^{-u/20}$	$1/(e^{3-0.1u}+1)$

Table 1.1: This table shows the functions related to the gating variables of the Hodgkin-Huxley model. Reproduced from [Gerstner 2002]

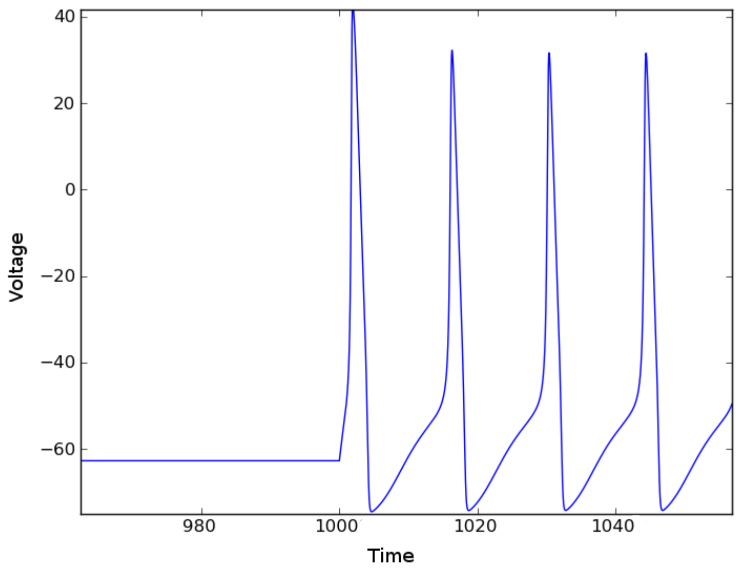


Figure 1.9: Example of the solution of the Hodgkin-Huxley equation. The input is 0 until time 1000, when an input value of 14 is applied. For more information see the text.

1.2.1.2 FitzHugh-Nagumo model

The Hodgkin-Huxley equations form a 4 dimensional system which is difficult to analyze and due to the difficulty to see more than a two dimensional projection it is hard to get an intuition of its solution. For this reason FitzHugh [FitzHugh 1955, Fitzhugh 1966, FitzHugh 1969] proposed a two dimensional reduction of the model that could keep the majority of the properties and reproduce interesting biological phenomena. As this model is composed of only 2 equations a geometrical analysis is possible. The equations for the model as presented in [Izhikevich 2006] are:

$$\begin{aligned} \dot{V} &= V - V^3/3 - W + I \\ \dot{W} &= 0.08(V + 0.7 - 0.8W), \end{aligned} \tag{1.5}$$

where V is the membrane potential, W is a recovery variable and I an external current. An example of the solution of this equation for a constant input can be seen in figure 1.10. This example shows the existence of spikes but of a different shape than the ones presented in figure 1.9 for the Hodgkin-Huxley model.

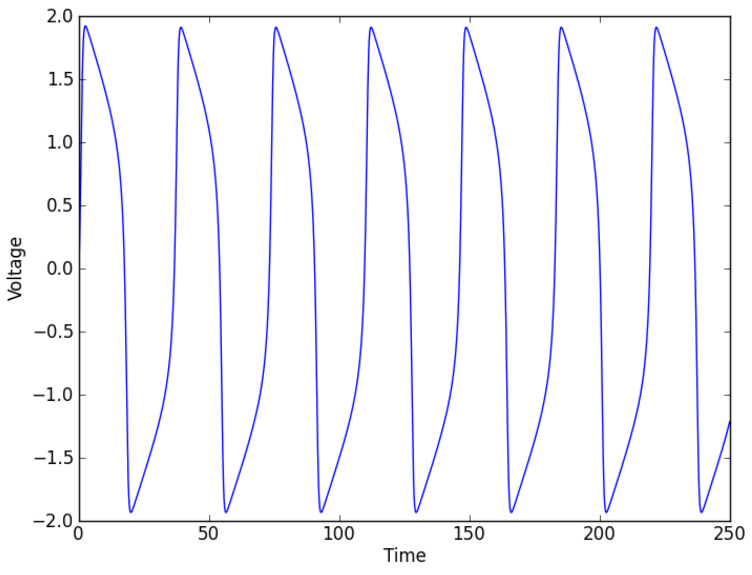


Figure 1.10: Example of the solution of the FitzHugh-Nagumo equation with input 0.7.

The model features one stable fixed point when I is small and an unstable limit cycle when I is higher. The fixed point produces a resting potential while the limit cycle produces a periodic activity which represents the emission of action potentials. For a complete bifurcation analysis of the model see [Izhikevich 2007].

1.2.1.3 Morris-Lecar model

Another two-dimensional reduction of the Hodgkin-Huxley equation is the Morris-Lecar model [Morris 1981]. In this case the Na+ channel is assumed to approach its asymptotic value extremely fast. This assumption is based on experimental data. The equations as presented in [Lecar 2007] are:

$$\begin{aligned} C\dot{V} &= -g_{Ca}M_{ss}(V)(V - E_{Ca}) - g_K W(V - E_K) - g_L(V - E_L) + I_{app} \\ \dot{W} &= (W_{SS}(V) - W)/T_W(V), \end{aligned} \tag{1.6}$$

where the conductance functions are given by:

$$\begin{aligned} M_{ss}(V) &= (1 + tanh[(V - V1)/V2])/2 \\ W_{ss}(V) &= (1 + tanh[(V - V3)/V4])/2. \end{aligned} \tag{1.7}$$

An example of the solution of this equation is presented in figure 1.11. The height of the spikes is much bigger than for the FitzHugh-Nagumo equations giving values closer to the more realistic Hodgkin-Huxley model. Another difference from the previous 2 dimensional reduction is the possibility to measure all of the parameters experimentally.

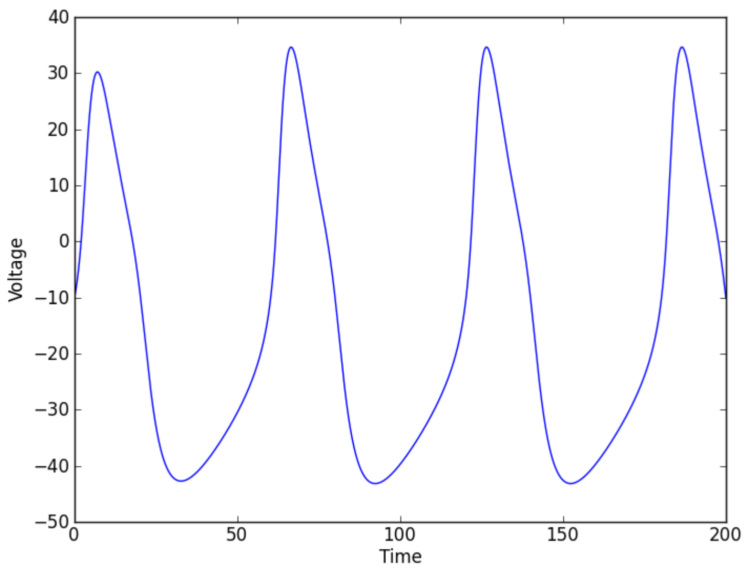


Figure 1.11: Example of the solution of the Morris-Lecar equation with input current 80.

For a complete bifurcation analysis of this model see [Izhikevich 2007].

1.2.1.4 Integrate and Fire models

In both previous models the shape of the spike was produced by the equation itself. In the case of the integrate and fire model only the behavior before the emission of an action potential is given, and a specific threshold is assigned to the neuron. If the membrane potential exceeds this threshold a spike is emitted and the voltage variable is reset to its resting potential.

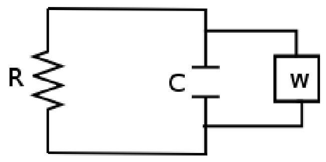


Figure 1.12: Circuit representing an integrate and fire neuron. A capacitor is connected with a resistance. Each time the threshold w is exceeded a pulse is emitted. Adapted from [Gerstner 2002].

The basic integrate and fire neuron can be seen as the simple circuit presented in figure 1.12. This circuit is a simplified version of the one in figure 1.8: it has a capacitor connected to just one resistor. An extra element is added that compares the voltage to the threshold and resets the potential if a spike is emitted. Using the same procedure as for the Hodgkin-Huxley model, the equation for the potential is:

$$I(t) = \frac{V(t)}{R} + C\dot{V}, \tag{1.8}$$

where R is the resistor conductance and C the capacitance. This equation is normally multiplied by RC to give:

$$\tau_m \dot{V} = -V(t) + RI(t), \tag{1.9}$$

where $\tau_m = RC$ is a time constant.

Figure 1.13 shows the evolution of the voltage for an integrate and fire neuron when the threshold is set at 1.0 and the input is 1.5. The main difference between this plot and the previous ones is that the shape of the spike is not seen, as an action potential is considered a discrete event that occurs when the threshold is reached.

Other more complex and realistic versions of this kind of model can be found in [Gerstner 2002].

1.2.2 Synapses

In order to understand correctly the collective activity of neuron assemblies we need not only comprehend the dynamics of a single cell but also how they interact with each other. The relationship between cells are the basis of the computations performed in the brain and occur at local synapses between neurons. It is through

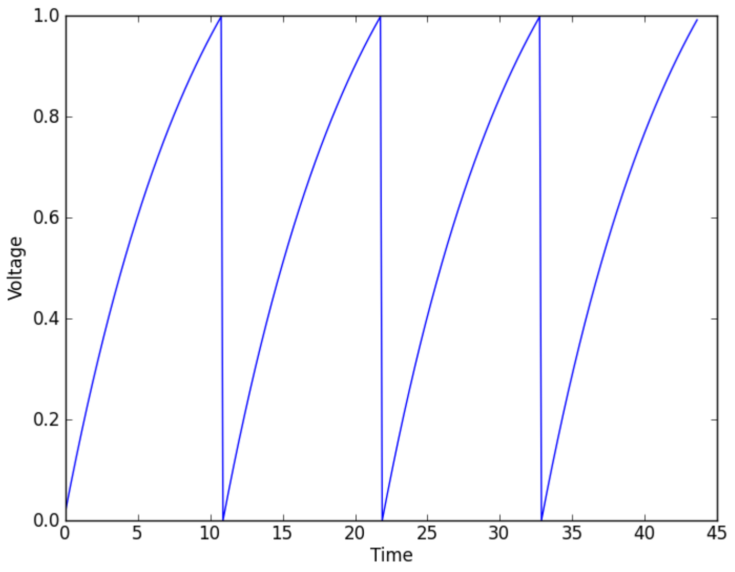


Figure 1.13: Solution of the integrate and fire equation. The threshold is set at 1.0.

the synapses that the spikes emitted by one cell modify the membrane potential of another.

In chemical synapses neurotransmitters are released by the presyanptic neuron that bind to receptors on the postsynaptic neuron. These receptors when activated cause the opening of synaptic channels that work in a similar way to other ion channels [Ermentrout 2010]. For this reason synaptic channels are modeled in agreement with the other currents entering the cell in the Hodgkin-Huxley model, i.e., with a product of a conductance and a voltage difference:

$$I_{syn}(t) = g_{syn}(t)(V - E_{syn}). \tag{1.10}$$

For excitatory synapses the reverse potential, E_{syn}, is 0 while for inhibitory synapses it is around -75mV [Gerstner 2002].The shape of the function $g_{syn}(t)$ depends on the kind of synapse modeled, but a common choice are alpha functions, as presented in [Ermentrout 2010]:

$$\begin{aligned} g_{syn}(t) &= \bar{g} \sum_k \alpha(t - t_k) \\ \alpha(t) &= \frac{a_d a_r}{a_r - a_d}(e^{-a_d t} - e^{-a_r t}), \end{aligned} \tag{1.11}$$

where t_k are the times at which the presynaptic cell has spiked, a_r and a_d are parameters that describe the rise and decay of the synaptic conductance.

The main disadvantage of this model is that it is necessary to save the time of each of the spikes emitted. Another option for modeling synapses is presented in

[Destexhe 1994, Ermentrout 2010] where the conductance is given by:

$$g_{syn}(t) = \bar{g}y(t), \tag{1.12}$$

where y(t) denotes the fraction of open ion channels. This function depends only on the membrane potential of the presynaptic neuron, V_{pre}, as:

$$\begin{aligned} \frac{dy}{dt} &= a_r S(V_{pre})(1-y) - a_d y \\ S(V_{pre}) &= \frac{T_{max}}{1+e^{-(V_{pre}-V_T)/Kp}}. \end{aligned} \tag{1.13}$$

The values proposed by the authors for the parameters are: $T_{max} = 1mM$, $V_T = 2mV$ and $Kp = 5mV$.

In electrical synapses (also called gap junctions), the 2 neurons are directly connected, without the gap where neurotransmitters are released. In this kind of union the 2 cells are always in communication and not only when a spike is emmited. This can be modeled in a similar way than before but with a conductance that does not vary with time:

$$I_{gap} = \bar{g}_{gap}(V_{post} - Vpre). \tag{1.14}$$

1.2.3 Mean field techniques

Neural networks in the cortex are composed of a big group of neurons and an even larger number of synapses. As each of these elements is described by several equations the complexity of the system is huge. As analyzing analytically or numerically a large scale network becomes untractable due to its number of components, several techniques have been applied to face this difficulty, mean field methods being one of them. In this kind of method the whole system of equations is reduced to a few of them that describe the mean behavior of the system when the number of neurons tends to infinity. Some important efforts in the application of these methods in neuroscience is presented here.

The application of mean field techniques in neuroscience is not new, and can be dated back to the seminal study of emergent behavior in continuum limits by Wilson and Cowan and Amari [Amari 1972, Amari 1977, Wilson 1972, Wilson 1973]. In these cases the equations derived describe a macroscopic characteristic of the network, like for example the mean firing rate, by a integro-differential equation. This kind of limit equation is usually called a neural mass or a neural field if it considers space.

Gerstner in [Gerstner 1995] proposed a mean field description for populations of neurons given by the spike response model. Each population is a dense network with low weights whose values depend only on the pool to which the postsynaptic and presynaptic neuron belongs. The equation derived describes the activity of the whole population instead of the activity of a single unit in each pool.

A mean field description for a network of integrate and fire neurons is proposed in [Brunel 1999]. In this case the connections in the network are sparse, a feature that produces uncorrelated activity between neurons. A reduced equation that describes the distribution of potentials in s population of neurons was obtained and used to understand the appearance of oscillatory solutions.

This approach was extended in [Mattia 2002] where an analysis of the solutions of the Fokker-Planck equation that describes the evolution of the probability density of the possible potential values is made. In this work the finite size effects are considered and the model is extended to a network with multiple populations.

Macroscopic equations based on the population density approach are obtained in [Chizhov 2007] for a network of more realistic neurons. In this case a simplification of a detailed current based model for a hippocampal pyramidal neuron is used. The authors replace the sodium current by a threshold function, obtaining a mechanism for the generation of action potentials similar to the integrate and fire model .

More recently, a different approach based on the development of a master equation was proposed in [ElBoustani 2009]. They have developed an equation that describes the evolution of the probability density that they can't solve exactly, so a moment expansion is used for further analysis. A truncation after the second moment is made, giving equations for the mean population activity and the covariance matrix.

A dynamic mean field approach is used in [Faugeras 2009]. In this case a network of neurons described by stochastic differential equations with random weights which depends only on the presynaptic and postsynaptic population is used. This is a more complex network topology than the previous approaches. A set of population activity equations are derived and proved to be well posed. Finally an algorithm for computing the solution to these equations is provided.

1.2.4 Neural field models of visual areas

1.2.4.1 General description

Another way to analyze large scale networks is to take the continuum limit and consider a macroscopic variable, like the mean firing rate, at every position. These kinds of models are called neural fields and have been studied since the works of Wilson and Cowan and Amari [Amari 1972, Amari 1977, Wilson 1972, Wilson 1973]. The main advantage of this approach is the development of simpler equations that can be treated analytically and numerically while its disadvantage is an inability to reflect the effects of the inter spike time. A general form for this kind of model, as described in [Coombes 2005, Bressloff 2012], is:

$$\frac{1}{\alpha}\frac{\partial V(x,t)}{\partial t} = -V + \int_{-\infty}^{\infty} w(y)S(V(x-y,t))dy, \tag{1.15}$$

where $V(x,t)$ is the activity of a population at position x, S is the firing rate function and $w(y)$ is the weight between elements separated by distance y. The w function is normally taken to be a Gaussian, an exponential or a mexican hat function that combines excitatory and inhibitory connections [Ermentrout 1998].

A typical choice for the firing rate function is a sigmoidal function of the form:

$$S(x) = \frac{1}{1 + e^{-\sigma(x-\theta)}}, \tag{1.16}$$

where σ is the nonlinear gain and θ is the threshold. When $\sigma \to \infty$, the function S becomes a Heavyside function, $H(u-\theta)$. In this case the neuron fires at its maximum rate or does not fire at all. The threshold determines the minimum potential needed for the generation of a spike.

Several biologically interesting phenomena have been studied with the use of this kind of model. Neural fields are able to maintain a pattern of activation even after the input has been removed, representing the capacity of the brain to keep information for a fast access even after the feature that triggered the activation has been removed from our senses. Several authors [Laing 2003a, Gutkin 2002, Laing 2003b, Rubin 2004, Guo 2005a, Guo 2005b] have shown the existence of sustained patterns of activity using different weight and firing rate functions. This may be used to explain working memory or short term memory.

Another phenomenon studied within this framework is the propagation of waves in cortical tissue. This kind of behavior has been reported in different areas of the cortex and can be observed with modern optic imaging techniques that measure a mesoscopic view of the activity [Lee 2005, Golomb 1997, Peinado 2000]. Neural fields also can show wave propagation as has been presented in [Coombes 2005, Bressloff 2001a, Kilpatrick 2008].

1.2.4.2 Models of the primary visual cortex

The primary visual cortex has been the subject of several modeling efforts as it is one of the areas of the cortex which has been studied experimentally the most. Several approaches have used the neural field techniques to explain some of the computation done in V1. One of the main objectives of using this method is to reduce the complexity of the complete network but to keep the properties that produce interesting phenomena. The columnar organization of V1 can be represented very naturally using a neural field approach as will be described later.

A neural field model for one hypercolumn of V1 known as The Ring Model of Orientation was introduced in [Hansel 1997]. As described earlier one hypercolumn groups several columns with different orientation preference but similar receptive field. In this case the space variable of the general model is used to represent the possible orientations, and a periodic weight function is used. The original equation for the model is:

$$\tau \dot{A}(x,t) = -A(x,t) + S\left[\int_{-\pi/2}^{\pi/2} J(x-y)A(y,t)dy/\pi + \varepsilon I(x)\right], \tag{1.17}$$

where A is the activity of the population with orientation preference x, J is a π periodic weight function and I represents the input coming from other areas of the brain. A change of variables of the form $V = \int_{-\pi/2}^{\pi/2} J(x-y)A(y,t)dy/\pi + \varepsilon I(x)$, gives an equation similar to the general case presented in (1.15):

$$\tau \dot{V} = -V + \int_{-\pi/2}^{\pi/2} J(x-y)S(V(y,t))dy + \varepsilon I(x). \tag{1.18}$$

Several different weight functions have been proposed for this model. In [Bressloff 2000, Bressloff 2001b] a difference of Gausian is used while in [Ben-Yishai 1995, Veltz 2011] a function of the following form is used:

$$J(x) = J_0 + J_1 \cos(2x). \tag{1.19}$$

The input function has the form:

$$I(x) = 1 - \beta + \beta \cos(2(x - x_0)). \tag{1.20}$$

This function has a weakly tuned shape with a peak at the angle x_0.

The model presents solutions that enhance the tuning of the input function, i.e., the function A has a sharper shape than I with a peak at the same angle. This kind of process improves the angle detection procedures that are done in the retina and the LGN. Figure 1.14 presents an example of this, on the left side the input function is shown and on the right the stationary solution of equation (1.17). The parameters for the simulation are given in the figure. For the input function presented is difficult to select an orientation as all of them have a similar activity. It is much easier to select an angle in the stationary solution as several orientations present no activity and the height of the peak is bigger.

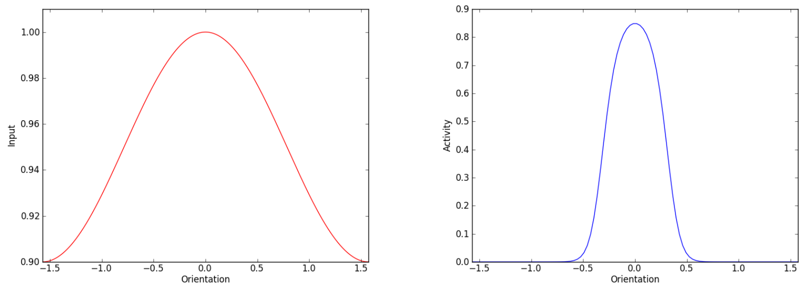


Figure 1.14: Left: input function for the ring model with β=0.05 and x_0=0. Right: stationary solution of the ring model with $\varepsilon = 0.01$, $\sigma = 23$ and $\theta = 2$

A previous work reported in [Veltz 2011] describes the conditions for the parameters in order for the proper tuning curve to exist. The setting of the nonlinear gain is critical as normally three solutions to 1.17 exist and two of them disappear at a bifurcation point leaving only the correct one. The author even shows the existence of an illusion as under certain condition an input with a peak at 0 can have an output tuning curve peaked at $\pi/2$.

Bresslof et. al. [Bressloff 2001b] used a spatial extension of the Ring Model to represent the complete primary visual cortex and to explain some visual hallucinations. In their model there is an infinite number of hypercolumns, one at each possible position, each represented by a Ring Model equation. They also add long range lateral connectivity between hypercolumns following biological constrains. The equation for the model is:

$$\frac{\partial V(r,\theta,t)}{\partial t} = -\alpha V(r,\theta,t) + \mu \int_0^\pi \int_{\mathbb{R}^2} w(r,\theta|r',\theta') S(V(r',\theta',t)) \frac{dr' d\theta'}{\pi} + I(r,\theta,t), \tag{1.21}$$

where α and μ are decay and coupling coefficients.

The weight function, w is the sum of a local part and a lateral or long range one. The local part is non-zero only on elements with the same position and has the same shape as the Ring model of connectivity. The lateral connectivity fulfills the following biological constrains: only elements of similar orientation preference are connected, the connections only join elements in the direction of the preferred orientation, they present short range excitation and long range inhibition. The final form of the weight function is:

$$w(r,\theta|r',\theta') = w_{loc}(\theta-\theta')\delta(r-r') + w_{lat}(r-r',\theta)\delta(\theta-\theta'), \tag{1.22}$$

with:

$$\begin{aligned} w_{lat} &= \hat{w}(R_\theta r) \\ \hat{w} &= \int_0^\infty g(s)[\delta(r-sr_0)+\delta(r+sr_0)]ds, \end{aligned} \tag{1.23}$$

where $r_0 = (1,0)$ and R_θ is the rotation of angle θ.

The authors make a bifurcation analysis with respect to the parameter μ that shows the existence of multiple solutions that represent well known visual hallucination when transformed from the retinotopical map to visual space coordinates. Some examples of the patterns they could compute are shown on figure 1.15. This analysis was extended in [Bressloff 2002b] where they show the effect of lateral connection on the shape of the tuning curves.

A different model is proposed in [Chossat 2009, Faye 2011], where each population in the primary visual cortex is considered to represent a structure tensor of the image, which contains information not only about edges but also about textures. They use the same general equation (1.17) but the spatial variable instead of representing just the orientation preference as in the Ring Model, represents a structure tensor. Although there is not enough biological evidence that columns in V1 use this kind of structure the authors provide enough theoretical analysis for experimenters to create a protocol to study this hypothesis.

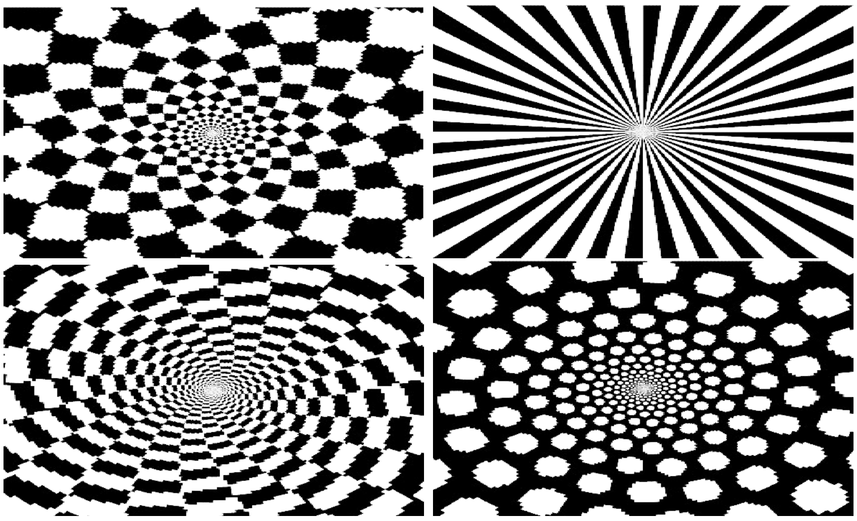


Figure 1.15: Hallucinations computed with the model of [Bressloff 2001b].

The structure tensor is a nonlinear representation of the image first derivative. It can be computed by first convolving the image with an isotropic or circulary symmetric Gaussian function with 0 mean and variance σ_1^2. Then the derivative at each point is computed for the new image. A 2x2 matrix is formed by applying a tensor product to the 2-dimensional vector resulting from the derivative computation by itself. Finally this matrix is convolved with a different Gaussian function with variance σ_2^2.

The parameters σ_1 and σ_2 represent the spatial scales. The first one, σ_1, indicates the minimum level of detail to which the structure tensor is sensitive. The second one, σ_2, is related to the size of the texture to be represented and to the size of the receptive field.

The distribution of the 2 eigenvalues of the structure tensor, λ_1 and λ_2, represent the organization of the intensities in the image. If an area has a constant intensity the 2 eigenvalues are 0, if a straight edge is present $\lambda_1 >> \lambda_2 \backsimeq 0$ and if a corner is present $\lambda_1 \geq \lambda_2 >> 0$. The difference between the eigenvalues becomes large for anisotropic textures.

1.3 High performance computing

High performance computing consists in the use of computers with a high number of processors (also called supercomputers) for solving complex problems that normally could not be solved on a personal computer. This kind of technique is useful when the problem consists of analyzing large volumes of data or when the algorithm needs to execute a big number of instructions. Not all large programs can be divided in different tasks to be made in parallel as normally there is a dependency between consecutive instructions.

Supercomputers started with Seymour Cray in the 1960s, who designed computers that were more powerful than any other at its time [Sisson 2006]. Since then the computational power and number of processor per machines have been increasing, and currently even a ranking has been created to keep track of the most powerful machines (TOP500). In fact, parallel computing techniques have become of more importance in the last decade, as the increase in the processor speed is reaching its limits [Lundstrom 2003] but the reduction in size has allowed the designers to incorporate more than one processor in just one chip. Nowadays even mobile phones that can be kept in a pocket have more than just one processor.

1.3.1 Architectures and programming paradigms

Several architectures have been proposed for the interconnection of different processors, each one providing a different way of programming the resulting machine. Some efforts have been made on normalizing the programming paradigm but normally at least the memory distribution needs to be accounted for. The main categories of current parallel machines and how they are programmed is described next.

The most well known way to characterize a parallel machine is by the use of the Flynn's taxonomy [Mattson 2004, Hennessy 2007, Flynn 1972]. In his approach there are 4 options: Single-instruction-single-data (SISD), Single-instruction-multiple-data (SIMD), Multiple-instruction-single-data (MISD) and Multiple-instruction-multiple-data (MIMD). A sequential computer is considered a SISD system, where one processor operates on one input stream. In a SIMD system each processor executes the same instruction but to a different data stream. In a MISD system multiple instruccions operate on a single data, this kind of architecture is hard to find in reality. In a MIMD system each processor executes a different instruction on its own data stream. This is the most general case and most modern computer clusters fit in this category.

Another way of classifying parallel machines is to divide them according to their memory structure [Mattson 2004, Hennessy 2007]. In a shared-memory environment there is just one single main memory for all the processing elements and on a distributed-memory environment each processor has its own private memory. The communication between processors is extremely different between the 2 modalities as in the first case each processor can read directly from the centralized memory data written by others while this is impossible in the distributed memory case. Distributed memory machines normally support a larger amount of processors and a bigger amount of total memory.

The easiest way to exploit the parallelism of a machine is to leave the compiler extract independent task from a sequential code [Kuck 2011]. The output code of the compiler will make the machine execute each of this task in parallel without the user managing explicitily the architecture. Some libraries, like OpenMP [Chapman 2008], allow the user to explicitly indicate which parts of the code must be parallelized by the compiler and give some guidance and how to treat them by setting some parameters in the code itself.

In some cases the code provided by a compiler is not enough to obtain the maximum profit from the parallel machine. The code generated may not be as efficient as it could be given the hardware constrains or the parallelization strategy is too complex for the compiler to handle it automatically. For these cases several programming models have been proposed but only 3 are the main cores of most of modern programming languages [Kuck 2011]. Message passing is the first model, where each process is independent and may send or receive data to or from others via packaged messages. The Message Passing Interface, MPI, is the most used standard for this approach. A complete description of MPI can be found in [Gropp 1994].

Another programming model is the Fork-join structure where any process may separate in 2 or more independent elements. These new processes or threads (depending on the library used) are executed separately and they may be joined or combined when finished. The communication is normally done using shared memory. A widely used example of this kind of model is the POSIX thread API, also called pthread, for UNIX type machines [Butenhof 1997]. This is a group of C types and functions that allow the programmer to create, join and synchronize threads.

A third model is used in the data parallel languages where a great deal of low level detail is expressed in the data itself. This is normally used in SIMD arquitectures where an array of values is updated in parallel. An example of this is GPU computing, for which more detail will be given later.

1.3.2 Current trends of Supercomputing

The amount of processors and cores on supercomputers is increasing every year. The current fastest supercomputer, as presented in the TOP500 list (November, 2012), has already 560,640 processor. Also, an important trend in the area is to start including new accelerators like GPUs (see 1.3.4) or FPGAs ([Sulaiman 2009]), that can work together with the processors to increase the speed. Already 13 of the first 100 supercomputers on the TOP500 list include GPUs and this number has been increasing in the last years. For a complete analysis of the November 2012 TOP500 list see [Deng 2013].

One of the most important problem in the design of current Supercomputers is energy consumption. As more processors are included more electrical power is required for them to work. The financial cost of this situation and the effect on the environment limit the possible size of machines. Currently, chips designers are moving to simpler cores that can work together to obtain a good performance but keeping the power consumption very low.

Another issue with this kind of machine is how to create software that is really able to use and take full advantage of this type of machine. Current parallel software should be able to scale to future machines with more and more processors. The problem is that the speedup of a program will still be limited by its sequential fraction and the comunication cost. This is normally called the Amdahl's law [Amdahl 1967].

Current analysis of supercomputers consider another way of scalability in which assumes that more powerful machines are created for more complex problems. For very large machines the time for solving the problem as a function of the number of processors is not measured with a fixed problem size (strong scaling) but for a fixed size per processor. This means that newer, more powerful machines, are thought to be used for larger problems. In many domains, models can be enhanced by adding more detail which makes the simulation more complex. More powerful machines will allow scientists to add this information to their equations and run simulations that approach realisitic figures.

1.3.3 HPC in neuroscience

High performance computing has been extensively used for the simulation of spiking neural networks. One of the main challenges of neural simulation is to deal with the large numbers of neurons or synapses. HPC provides an option for facing this kind of problem by the uses of its larger computational power and memory. Simulations can approach real biological figures when a large computer cluster is used.

Several softwares have been created for the simulation of spiking neural network. Some of them, like NEST [Gewaltig 2007] or NEURON [Hines 2002], are designed to take advantage of computer clusters when available. For a review on the different available tools see [Brette 2007]. The general approach is that the user expresses the network characteristics independently of the machine where the simulation is going to take place and then the simulator automatically deals with the parallelization.

Numerous new problems arise when moving from a sequential simulation to a parallel one. The first problem is to determine the distribution of neurons and synapses between the different processors. Depending on the organization, synapses may be between cells on different machines, increasing the amount of communication needed for the simulation. The data structures representing the network topology also need to consider this fact.

A second problem is to create an efficient communication scheme to spread the spike information. It is necessary to minimize the number of exchanges as they have a large influence on the total computational time. Coordination is also important as spikes may be lost due to a bad communication strategy.

The solution of these and other problems depends on the simulator. The algorithms used in the software NEST are presented in [Morrison 2005]. This software is specially designed to run spiking neuron simulations on large computational clusters. The solutions the authors propose were tested on different clusters and show a linear increase in performance when increasing the number of processors.

A different view on the use of HPC in neuroscience is presented in [Hines 2008]. The authors do not distribute a neural network among the processors but do so for just one neuron described by a very detailed model. The model considers the cell real shape and divides it into small compartments, each described by a different equation. In this approach different cell sections are connected together and it is this dependency that must be considered for the communication strategy. As such, the load balancing problem still exists. An implementation is available on the software

NEURON, where even a network of compartmental cells can be simulated in parallel. Tests show an ideal scaling with the number of processors.

To our knowledge the majority of the efforts in the use of high performance computing in neuroscience do not deal with mesoscopic or macroscopic models but only with spiking neurons. This is probably due to the fact that normally the reduction of complexity given by the simplified population equations is enough as to solve the system in a sequential computer. We will show in this thesis that this is not true when the population equations for a more realistic model are derived. Also we will show that when a very large piece of the cortex needs to be simulated a mixture of mesoscopic modeling and parallel computing is extremely useful.

1.3.4 GPU computing

Graphic processing units, GPUs, where introduced in the 90s to improve real time graphic performance for games. They were originally designed for fast floating point arithmetic to calculate 3D geometry and update pixel values [Nickolls 2010]. As the years passed the technology evolved and currently GPUs have become more flexible and are not only used for games but also for scientific applications. See [Garland 2008] for a review on common applications on science and engineering.

Modern GPUs are low cost highly parallel devices that present a cheaper and powerful option to computer clusters and standard high performance computing solutions. Any workstation can be transformed into s small supercomputer by adding one of these cards. Also any supercomputer may beenfit from this technology by connecting several GPUs in parallel to work together with its standard processors [Kindratenko 2009]. In fact, as of November 2012, the most powerful supercomputer in the word, as selected in the TOP500 list, and several others in the first 10 use GPUs by nVidia to power up their computation.

The core of modern GPUs are a set of streaming multiprocesors that work in a SIMD manner [Owens 2008]. Each processor executes the same set of instructions on different pixels values, in the case of graphics applications, or over different array values in the case of scientific applications. This first constraint limit the variety of problems for which GPUs are useful, but as will be showed later, several neuroscience applications are well suited for this kind of hardware.

There are mainly 2 companies that produce GPUs that can be used for general computation, nVidia and ATI [Owens 2008]. In order to describe in detail the architecture of a GPU this document will focus on the nVidia cards (Fermi architecture) and their programming model, as it is the one used for the experiments presented later. For more information about ATI cards see [Bayoumi 2009] and on the programming paradigm used for them, called OpenCl, see [Stone 2010]. OpenCl is an interesting approach to the programming of parallel devices as it is independent of the architecture, the language is designed for the same code to run on any multiprocessor machine, like GPUs.

A block diagram of the nVidia Fermi architecture for GPUs ([Glaskowsky 2009, Wittenbrink 2011]) is presented in figure 1.16. The green elements of the figure are a set of streaming multiprocessors, each consisting of 32 cores. The GigaThread element, in orange, is in charge of scheduling. The blue elements are in charge of accessing DRAM and a L2 cache is included. The majority of the space is used for the computational elements which may execute a huge amount of threads in parallel. As mentioned in [Glaskowsky 2009], this is the main difference with Intel type processors, where most of the space is dedicated to speculation on the next possible instruction to be needed.

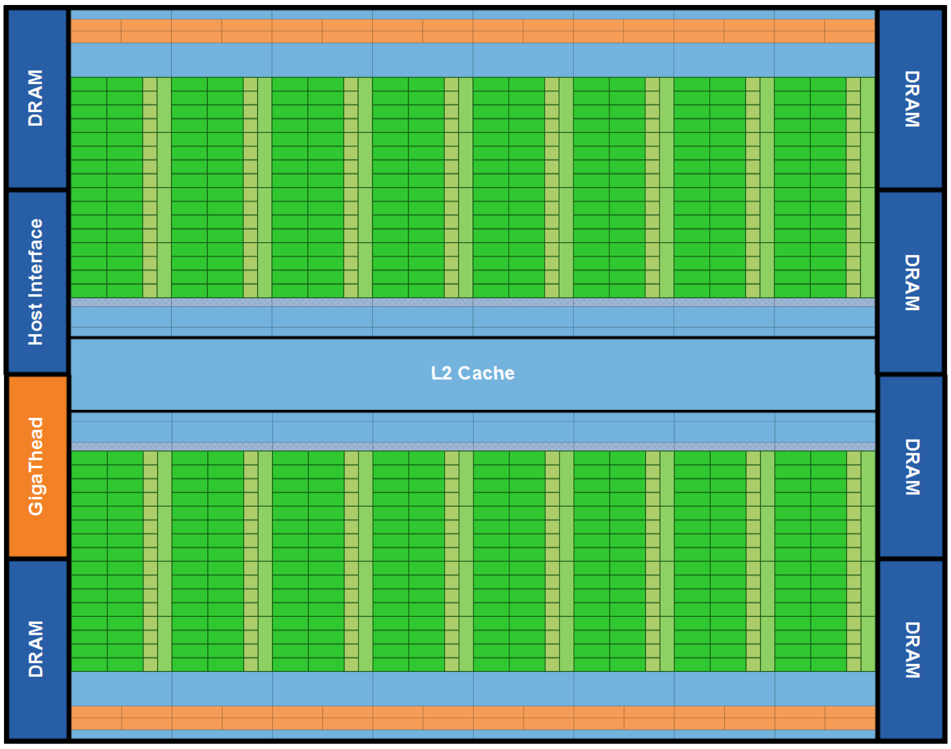


Figure 1.16: Block diagram of the nVidia Fermi architecture. Taken from [Glaskowsky 2009]. More details in text.

A block diagram for a single streaming multiprocessor is presented in figure 1.17. Each of them includes 32 cores, as indicated in green in the diagram, these can execute one single precision operation per clock period. A set of 16 load store units (LD/ST) are in charge of memory operations. These can handle addresses in term of 2 dimensional arrays and perform format conversions. Special Function Units (SFU) are available to handle special operations like sin, cos or exp.

Threads are divided in a three-dimensional grid of blocks and then joined in groups of 32, called warps. This gives the maximum number of instructions that can be executed in parallel. The cores are divided in 2 groups of 16, called execution blocks. The warp scheduler and the dispatch unit are in charge of assigning the instructions either to one of the 2 execution blocks, to the load/store units or to the SFUs. A warp of special functions takes eight cycles to complete on the four

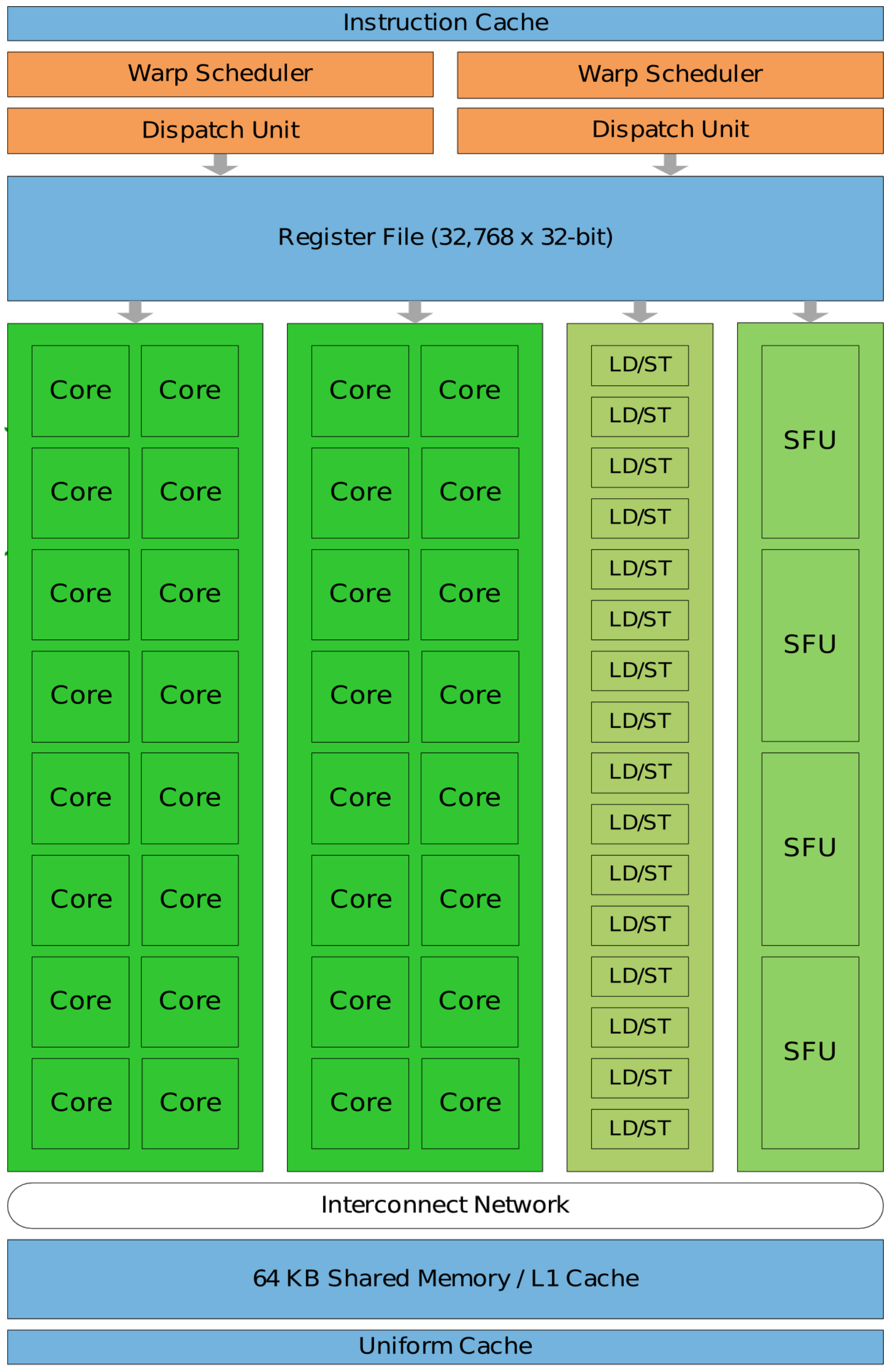


Figure 1.17: Block diagram of the nVidia multiprocessor. Taken from [Glaskowsky 2009]. More details in text.

available SFUs while a normal one takes 2.

A small local memory is present on each multiprocessor. This very fast memory may be managed by the programmer (shared memory) or left to work automatically as cache. Shared memory can be accessed by all the threads in a block and it is normally used for storing intermediate results or moderate amounts of common data. The efficiency of an algorithm may depend severely on its shared memory use as one access may be several orders of magnitude faster than a fetch from global memory.

CUDA is the programming model that nVidia has created to deal with their graphic processing units. It is available as a set of libraries and extensions to standard C/C++. The company provides a special compiler for this kind of code called nvcc. The core of the system is the separation of host code, to be run on a standard Intel type processor, and a device code to be executed in the gpu.

The functions written for execution on the device are called kernels. The kernels span a large amount of threads and the same code is executed on all of them. A typical example is a matrix times matrix multiplication where one thread is created for each element of the resulting matrix [Kirk 2010]. The programmer must create a three dimensional grid of thread blocks before the execution of any kernel. The limit for the shape of this grid depends on the card to be used but it is normally recommended not to use the maximum but a combination that maintains the multiprocessors completely occupied (generally a multiple of the numbers of cores, 32 on Fermi).

In CUDA the programmer must manage explicitly the host memory and the device memory. As both elements are separate units they can't access other's memory directly. The language provides primitives for allocating space in GPUs global memory and copying data from and to it. If an algorithm is input-output intensive this may become a bottleneck as most of the time may be spent sending data. Well suited applications for GPU computing normally are computationally intensive tasks where only a few memory transfers between host and device are performed.

There have already been some efforts on running spiking neural networks simulations on GPUs. Some works in this direction were done even before CUDA appeared [Bernhard 2006]. Nemo [Fidjeland 2009] is an open source simulator developed for the simulation of Izhikevich type neurons on CUDA-enabled GPUs. They use a similar approach to the one proposed in [Nageswaran 2009]. In both cases one of their main problems is to create data structures that can maintain a coalesced access when reading the synapses values.

A spiking network designed for an image processing task is used a benchmark to compare the performance of GPUs vs other multiprocessors architectures in [Mohammad A. Bhuiyan 2010]. Their results show that when the Hodgkin-Huxley model is used GPUs present the greatest speed while for the simpler Izhikevich model an Intel Xeon processor is the fastest. The amount of operations done for updating one Izhikevich neuron is much smaller that for the Hodgkin-Huxley, making the task a less computationally demanding problem, less suited for GPUs. This condition probably changes if the network is big enough but the authors do not study this effect. Although their results depend heavily on the optimization techniques used in each case they show an expected behaviour given the constraints of GPU computing.

Chapter 2

Numerical analysis of large scale neural networks using mean field techniques

Contents

2.1 A mean field reduction for conductance-based neurons

All the mean field techniques described earlier use a neuron model that includes a threshold for the generation of action potentials. The equations that describe a cell in these approaches are simpler than the more realistic Hodgkin-Huxley or its 2 dimensional reductions. Due to this difference in complexity a mean field description of a network of conductance-based models requires different mathematical techniques.

In this section a new approach for the derivation of mean field equations for a noisy version of a network of conductance-based neurons is presented as it has been published in [Baladron 2012b]. A detailed description of all the proofs will not be given as this falls outside the scope of this thesis, only the necessary information to understand the numerical analysis that is presented later will be given.

2.1.1 Noisy network model

The original derivation of current based models presented in section 1.2.1 do not consider the stochastic nature of the environment or of the channels that are present in the membrane of the cells. This can be incorporated, in the case of the Hodgkin-Huxley model, by adding a stochastic part to the external current and by modifying the equations for the gating variables. Similar changes can be made to the other models. The stochastic networks considered in this thesis will be described next.

The equations (1.4) for the gating variables of the Hodgkin-Huxley model are obtained when the proportion of open channels is computed using a Markov chain model. The process obtained can be shown to converge to equations (1.4) considering a infinite number of channels and other standard assumptions [Pakdaman 2010, Goldwyn 2011]. A more realistic approach taking into account the finite number of channels through the Langevin approximation results in the following stochastic differential equations [Wainrib 2010]:

$$dx = (\alpha_x(V)(1-x) - \beta_x(V)\,x)\,dt + \sqrt{\alpha_x(V)(1-x) + \beta_x(V)\,x}\;\chi(x)\,dW_t^x, \quad (2.1)$$

where $x = \{m, n, h\}$, W_t^x are independent standard Brownian motions and $\chi(x)$ is a function that vanishes outside (0,1). This guarantees that the solution stays between 0 and 1 for all times.

By considering an external current with a stochastic and a deterministic part the equation for the membrane potential of the Hodgkin-Huxley model are transformed into the following stochastic differential equations:

$$CdV_t = \Big(I(t) - g_K n^4(V_t - E_K) - g_{Na} m^3 h(V_t - E_{Na}) - g_{Cl}(V_t - E_{Cl})\Big)\,dt + \sigma_{ext}\,dW_t. \quad (2.2)$$

The same change to the external current can be made to the FitzHugh-Nagumo model giving the following general stochastic differential equations:

$$\begin{aligned} dV_t &= (V_t - \frac{V_t^3}{3} - w_t + I(t))\,dt + \sigma_{\text{ext}}\,dW_t \\ dw_t &= c\,(V_t + a - bw_t)\,dt, \end{aligned} \quad (2.3)$$

Figure 2.1 gives an example of the evolution of the membrane potential for the 2 models. These were obtained using the Euler-Maruyama method [Mao 2007]. In both cases the shape of the voltage is different to the previous examples concerning the deterministic models (see figure 1.9 for Hodgkin-Huxley and figure 1.10 for the FitzHugh-Nagumo) but action potentials are still present.

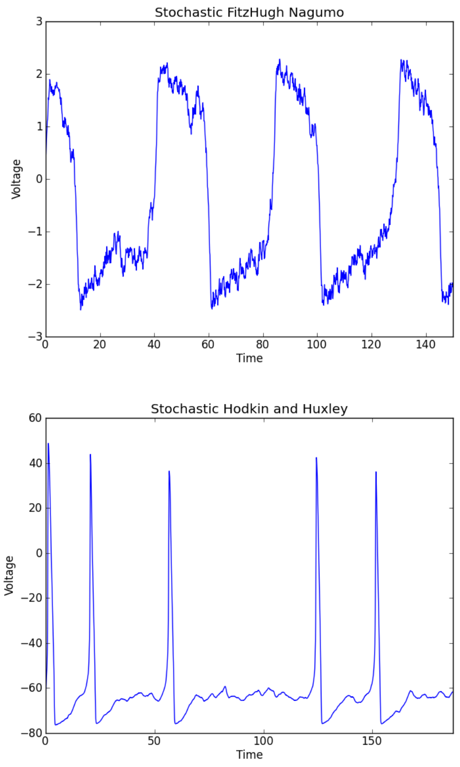


Figure 2.1: Examples of the solution to the stochastic models (2.3) and (2.2). Top FitzHugh-Nagumo, bottom Hodgkin-Huxley. For the FitzHugh-Nagumo the external noise intensity (σ_{ext}) is 0.27 and the deterministic input is constant and 0.7. For the Hodgkin-Huxley the external noise intensity (σ_{ext}) is 0.1 and the deterministic input is constant and equal to 3.0

For the synapses we consider the model described by equations (1.13) with a variable, y, that describes the fraction of open synaptic channels on each neuron. The maximum conductance, $\bar{g}$, is considered to depend on time and on the presynaptic and postsynaptic neuron populations. These values will also be called the weights of the network. The stochasticity of the synaptic channels is considered by modifying the equation in the same way that was done for the gating variables. Finally, the evolution of the variable y is given by the following stochastic differential equation:

$$dy_t^j = \left(a_r^\gamma S_\gamma(V_t^j)(1-y_t^j) - a_d^\gamma y_t^j\right) dt + \sigma_\gamma^y(V_t^j, y_t^j)\, dW_t^{j,y}, \tag{2.4}$$

where j is the neuron index (ranging from 1 to the total number of neurons, N), γ is the population of neuron j (it varies between 1 and P), $W^{j,y}$ are independent standard Brownian motions and the function σ is given by:

$$\sigma_\gamma^y(V^j, y^j) = \sqrt{a_r^\gamma S_\gamma(V^j)(1-y^j) + a_d^\gamma y^j}\chi(y^j). \tag{2.5}$$

The maximum conductances are considered to be independent diffusion processes with mean $\frac{\bar{J}_{\phi\gamma}}{N_\gamma}$ and standard deviation $\sigma_{\phi\gamma}^J$, where N_γ is the amount of neurons in population γ. This can be expressed in the following equation:

$$J_{i\gamma}(t) = \frac{\bar{J}_{\phi\gamma}}{N_\gamma} + \frac{\sigma_{\phi\gamma}^J}{N_\gamma}\xi^{i,\gamma}(t), \tag{2.6}$$

where ϕ is the population of neuron i and the $\xi^{i,\gamma}(t)$, $i = 1, \ldots, N$, $\gamma = 1, \ldots, P$, are independent zero mean unit variance white noise processes.

The main disadvantage of this approach is that if the noise level $\sigma_{\phi\gamma}^J$ is increased the probability that $J_{\phi\gamma}$ becomes negative also increases. To solve this problem the dynamics proposed in [Cox 1985] can be used:

$$dJ_{ij}(t) = \theta_{\phi\gamma}\left(\frac{\bar{J}_{\phi\gamma}}{N_\gamma} - J_{ij}(t)\right)dt + \frac{\sigma_{\phi\gamma}^J}{N_\gamma}\sqrt{J_{ij}(t)}dB^{i,\gamma}(t). \tag{2.7}$$

If the initial value is positive and the condition $2N_\gamma\theta_{\alpha\gamma}\bar{J}_{\alpha\gamma} \geq (\sigma_{\alpha\gamma}^J)^2$ holds the process is guaranteed to be strictly greater than zero [Cox 1985].

Finally, if we collect the synapses and the neurons equations, a network of N Hodgkin-Huxley type neurons is described by the following 5N stochastic differential equations:

$$\begin{cases} CdV_t^i &= \left(I^\phi(t) - \bar{g_K} n_i^4(V_t^i - E_K) - \bar{g_{Na}} m_i^3 h_i(V_t^i - E_{Na}) - \bar{g_L}(V_t^i - E_L)\right) dt - \\ & \quad \left(\sum_{\gamma=1}^P \frac{1}{N_\gamma} \sum_{j,\, p(j)=\gamma} \bar{J}_{\phi\gamma}(V_t^i - V_{\text{rev}}^{\phi\gamma})y_t^j\right) dt - \\ & \quad \sum_{\gamma=1}^P \frac{1}{N_\gamma}\left(\sum_{j,\, p(j)=\gamma} \sigma_{\phi\gamma}^J (V_t^i - V_{\text{rev}}^{\phi\gamma})y_t^j\right) dB_t^{i,\gamma} + \\ & \quad \sigma_{\text{ext}}^\phi\, dW_t^i \\ dx_t^i &= (\alpha_x^\phi(V^i)(1-x_t^i) - \beta_x(V^i)x_t^i)\, dt + \sigma_x(V^i, x_t^i)dW_t^{x,i} \quad x \in \{n,\, m,\, h\} \\ dy_t^i &= \left(a_r^\phi S_\phi(V_t^i)(1-y_t^i) - a_d^\phi y_t^i\right) dt + \sigma_\phi^y(V_t^i, y_t^i)dW_t^{i,y}, \end{cases} \tag{2.8}$$

where $p(j)$ is the population of neuron j, P is the total number of populations and ϕ is the population of neuron i.

In the case of the FitzHugh-Nagumo model it is described by the following 3N stochastic differential equations:

$$\begin{cases} dV_t^i &= \left(V_t^i - \frac{(V_t^i)^3}{3} - w_t^i + I^\phi(t)\right) dt - \\ & \quad \left(\sum_{\gamma=1}^P \frac{1}{N_\gamma} \sum_{j,\, p(j)=\gamma} \bar{J}_{\phi\gamma}(V_t^i - V_{\text{rev}}^{\phi\gamma}) y_t^j\right) dt - \\ & \quad \sum_{\gamma=1}^P \frac{1}{N_\gamma} \left(\sum_{j,\, p(j)=\gamma} \sigma_{\phi\gamma}^J (V_t^i - V_{\text{rev}}^{\phi\gamma}) y_t^j\right) dB_t^{i,\gamma} + \\ & \qquad \sigma_{\text{ext}}^\phi \, dW_t^i \\ dw_t^i &= c_\phi \left(V_t^i + a_\phi - b_\phi w_t^i\right) dt + \sigma_w dW_t^{i,w} \\ dy_t^i &= \left(a_r^\phi S_\phi(V_t^i)(1-y_t^i) - a_d^\phi y_t^i\right) dt + \sigma_\phi^y(V_t^i, y_t^i) dW_t^{i,y}. \end{cases} \tag{2.9}$$

2.1.2 Mean field description

It is proven in [Baladron 2012b] that this kind of network presents the propagation of chaos property which shows that when the initial conditions are independent and the number of neurons tends to infinity the cells become independent. This is only possible if the network has a tight interconnectivity and many independent sources of noise. The biological experiments in [Ecker 2010] agree with this theory.

Since, thanks to the propagation of chaos all the neurons become independent, the dynamic of each individual neuron is described by the same stochastic process. This process is not described by $3N$ or $5N$ equations as the complete network but by just $3P$ or $5P$ depending on the neuron model. The evolution of each cell can be understood as a sample of this process.

This new process that describes the law of each neuron can be obtained by changing the sum over all the connections present on the network equation by the mean value of y. This approximation is only accurate if the network has a dense connectivity. For the FitzHugh-Nagumo model the behavior of the neurons is ruled by the following 3 stochastic differential equations:

$$\begin{cases} dV_t^\phi &= \left(V_t^\phi - \frac{(V_t^\phi)^3}{3} - w_t^\phi + I^\phi(t)\right) dt - \\ & \quad \left(\sum_{\gamma=1}^P \bar{J}_{\phi\gamma}(V_t^\phi - V_{\text{rev}}^{\phi\gamma}) \int y P_\gamma(V,w,y,t) dV dw dy\right) dt - \\ & \quad \sum_{\gamma=1}^P \left(\sigma_{\phi\gamma}^J (V_t^\phi - V_{\text{rev}}^{\phi\gamma}) \int y P_\gamma(V,w,y,t) dV dw dy\right) dB_t^{\phi,\gamma} + \\ & \qquad \sigma_{\text{ext}}^\phi \, dW_t^i \\ dw_t^i &= c_\phi \left(V_t^i + a_\phi - b_\phi w_t^i\right) dt + \sigma_w dW_t^{i,w} \\ dy_t^i &= \left(a_r^\phi S_\phi(V_t^i)(1-y_t^i) - a_d^\phi y_t^i\right) dt + \sigma_\phi^y(V_t^i, y_t^i) dW_t^{i,y}, \end{cases} \tag{2.10}$$

where $P_\gamma(V,w,y,t)$ is the probability density function, for the possible states of a neuron in population γ at time t. A similar approach can be used to obtain the mean field equations for the Hodgkin-Huxley model.

Equations (2.10) can't be solved unless the values of $P_\gamma(V,w,y,t)$ are known. These quantities can't be computed beforehand unless the network equations are solved. One option is to numerically simulate the network in a Monte Carlo fashion to produce samples and then use them to approximate the probability density. These seems to contradict the original objective of the mean field approach to reduce the complexity of the system.

Another approach is to transform the mean field equation into a Fokker-Planck Equation that describes the evolution of the probability density function itself. This is a partial differential equation (PDE), which is completely deterministic, for which no previous computations are needed as the probability density is the unknown. In [Baladron 2012b] it is proven that the mean field equations described earlier can be expressed as its equivalent Fokker-Planck equation. For the case of equation (2.10) the corresponding Fokker-Planck equation is:

$$\begin{aligned}\partial_t P_\phi(t,V,w,y) = \sum_{\gamma=1}^{P} \frac{1}{2}(\sigma^J_{\phi\gamma})^2 \bar{y}^2_\gamma(t) \frac{\partial^2}{\partial V^2}\Big[(V - V^{\phi\gamma}_{rev})^2 P_\phi(t,V,w,y)\Big] + \\ \frac{1}{2}\frac{\partial^2}{\partial y^2}[\sigma^2_Y(V,y)P_\phi(t,V,w,y)] + \frac{1}{2}\sigma^2_{ext}\frac{\partial^2}{\partial V^2}[P_\phi(t,V,w,y)] + \frac{1}{2}\sigma^2_w \frac{\partial^2}{\partial w^2}[P_\phi(t,V,w,y)] \\ - \frac{\partial}{\partial V}\left[\left(V - \frac{V^3}{3} - w + I_{ext}(t) - \sum_{\gamma=1}^{P} \bar{J}_{\phi\gamma}(V - V^{\phi\gamma}_{rev})\bar{y}_\gamma(t)\right) P_\phi(t,V,w,y)\right] \\ - \frac{\partial}{\partial w}\left[a(V + b - cw)P_\phi(t,V,w,y)\right] - \frac{\partial}{\partial y}\left[(\alpha_r S(V)(1-y) - \alpha_d y)P_\phi(t,V,w,y)\right], \end{aligned} \tag{2.11}$$

where $\bar{y_\gamma}(t) = \int y P_\gamma(t,V,w,y) dV dw dy$.

Equation (2.11) is a 4 dimensional nonlinear and nonlocal partial differential equation. The integral term differentiates this from most other Fokker-Planck formalism where simpler equations are normally obtained. This equation can be solved with numerical methods designed for solving PDE as will be shown later.

The network of Hodgkin-Huxley neurons can also be reduced to its mean field equation and then to its Fokker-Planck reduction obtaining a 6 dimensional PDE. This high dimensionality is a challenge for current numerical methods and computers. The equations are not written here as they are not used afterwards in this thesis and would require an enormous amount of space.

By computing the solution to the Fokker-Planck equation all the dynamics of the network can be obtained. All the possible statistics from the network, like for example the mean firing rate, can be computed using this solution. Also samples from the process can be obtained from the probability density, representing voltage traces of neurons in the network.

2.1.3 Hardware setup

We have used a GPU cluster to run all the simulations presented in this thesis, getting an amazing increase in speed as has been reported in [Baladron 2012a]. The hardware is composed of 2 machines, each one with 7 nVidia Tesla cards. The cards in the 2 computers are different, one being the Tesla C2070 and the other the C2050

Number of CUDA cores	448
Frequency of CUDA cores	1.15GHz
Double precision floating point performance	515 Gflops
Single precision floating point performance	1.03 Tflops
Memory Speed	1.5Ghz

Table 2.1: Detailed information about the GPUs

Processor type	Xeon 2665
Number of processors	2
Number of cores per processor	6
Clock speed	2.66 GHz
Cache memory	12M
Number of PCI slots	7
Number of Tesla cards	7
PCI express version	gen2 16x interface
Memory Type	ECC DDR3 1333Mhz
Amount of memory	72Gb
Max memory bandwith	32GB/s
Network connection	infiniband QDR (32 Gbit/s)

Table 2.2: Hardware capabilities of each of the 2 available machines

cards. For detailed information about the cards see table 2.1. The only difference between the 2 is the amount of available memory. A block diagram of the cluster is presented in figure 2.2. The yellow squares in the figure show that there is at least one processor per card and that the communication between them can be done through shared memory, if they are in the same machine, or via a high speed network connection, if they are on different computers.

The computer has 2 Intel dual-Xeon X5650 processors, each one composed of 6 cores running at 2.67 GHz. More information about the hardware can be found in table 2.2

Using this kind of hardware requires the synchronization of 3 different levels of parallelism. The first one is inside each card, where multiple threads run concurrently. The second is inside each computer, where the different processor/GPU pairs must be coordinated. Finally, the third level is between the 2 computers that must work together and not independently of each other. Any software must provide a way to manage each of these levels.

2.1.4 Propagation of chaos in the Hodgkin-Huxley network

The propagation of chaos property shows that when the number of neurons tends to infinity they should become independent. This can be measured numerically by solving the network equations and measuring the correlations between any pair of

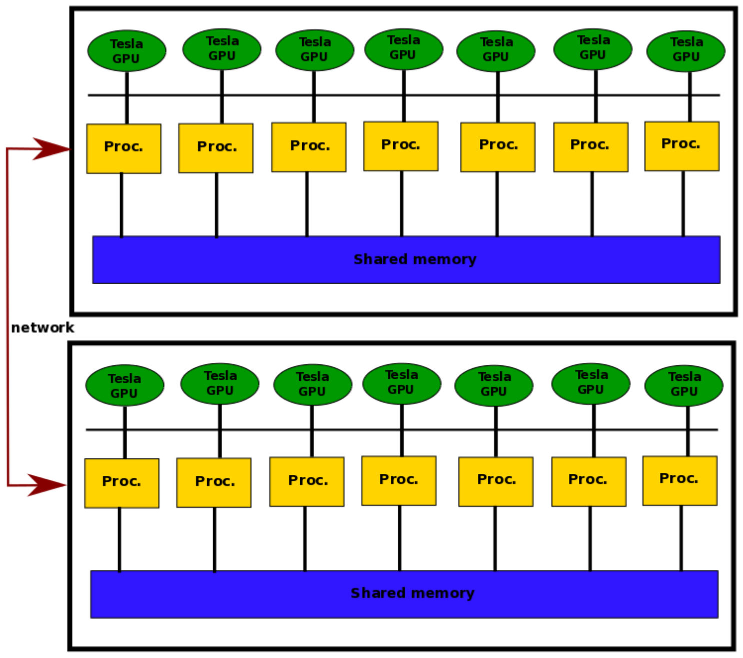


Figure 2.2: Block diagram of the GPU cluster on which all the numerical simulation presented in this document were performed.

elements. We expect this value tp be large for small network and to diminuish as the size increases.

In order to see the propagation of chaos in a network of the Hodgkin-Huxley type, we solve the network equations M times for $0 \leq t < T$ (of course, we discretize time) and for each simulation j we keep track of the trajectories of 2 randomly chosen neurons, i_1 and i_2. Then the following quantity was computed:

$$Corr_{i_1,i_2}(t) = \frac{1}{M}\sum_{j=1}^{M} \frac{V_j^{i_1}(t) - \bar{V}^{i_1}(t)}{\sigma_{i_1}(t)} \times \frac{V_j^{i_2}(t) - \bar{V}^{i_2}(t)}{\sigma_{i_2}(t)}. \qquad (2.12)$$

This is an estimate of the correlation between the membrane potential values of the neurons i_1 and i_2. This procedure was repeated for different network sizes.

As the standard deviation in Monte-Carlo simulations is of the order of $\frac{1}{\sqrt{M}}$, we need large values of M (beside large values of N) to test our hypotheses. In order to see the real quantities the magnitude of the error must be smaller than the correlation value. As this last number should be small for large N, the number of simulations required is enormous transforming this problem into a big computational challenge.

Another computational challenge in this kind of simulation is the generation of uncorrelated random numbers. The Brownians shown in equations (2.8) are assumed to be independent, and any correlation between them may change the results. We are approximating the solution of the equations by the Euler-Maruyama method [Mao 2007] which requires at each time step the generation of 6N uncorrelated random numbers. For large N and long simulations, as in our case, this becomes an extremely hard task.

A completely connected network is the simpler topology for which the dense connectivity required by the propagation of chaos effect is fulfilled. The simulations we have done use this kind of structure which also allows us to avoid the construction of complex data structures to handle synapses. This is one of the main problems faced by the kind of software tool described in section 1.3.3. In our case we just need to sum over the y variables of all the cells.

The generation of random numbers in parallel is not simple and is the bottleneck of any Monte Carlo method, including ours [Srinivasana 2003]. The simplest way to do this is to make each simulation in a different processor using multiple instances of the same sequential generator. In this case each process would run without communicating with the others until all the simulations assigned to it are finished. Then, once all the process are terminated, another step is necessary to join the results from all the simulations. This is an interesting approach as it has no communication delays. The problem is that correlation between the random numbers should be 0, independent of the processor where they were generated or the simulation for which they were used. Normally this is a difficult constraint to satisfy as the generated values depends on the seed assigned to each generator. If all the seeds are the same, the numbers are all the same, while if they are different, the selection may cause undesired correlations. As the original algorithm is designed to work sequentially it can't assure that independence will happen between sequences with different seeds. The problem of choosing a correct set of seeds becomes harder as the number of processors increases and tests are required to determine the existence of correlations between streams [Coddington 1998]. Another approach is to design random number generator algorithms that are adapted to distributed environments and that can use just one seed for all the processes [Jeng 2000, Ackermann 2001].

The most useful tool for solving this problem in our simulations is the Curand library develop by nVidia for its graphic cards, and that can be used in CUDA programs. This library allows us to generate a huge amount of high quality uncorrelated random numbers in parallel by providing just one seed. The results are stored inside the card and can afterwards be used to compute the righthand side of equation (2.8), as required by the integration scheme, without any memory transfer to the CPU.

The version of Curand we have used contains an implementation of a XORWOW type random number generator, which is very well suited for GPUs. This algorithm is different from the standard algorithm used for the sequential generation of high quality random numbers, called the Mersenne Twister [Makoto 1998]. When our implementation was written the only available implementation of Mersenne Twister for GPUs was an example provided in the Cuda Software Development Kit. This version used a simplification of the original algorithm with a reduced state which was small enough to be used with the small amount of memory of the GPU. Because of this change, some statistical tests were not passed by the algorithm, reason why it was not included in Curand. The current version of Curand (released after the development of our implementation but before the final version of this document was finished) includes a newer version of Mersenne Twister. This is not the same algorithm as in the Software Development Kit, but an improved version. A possible upgrade to our code should consider changing the random number generator to this new version of Mersenne Twister, this should reduce even more the correlation.

To test the quality of the version of the Curand random number generator we have used, we measured the correlation between the random numbers used for solving the voltage equation on a simulation of a 1,000 neuron network. We recorded the values that were generated for 2 randomly chosen cells at each of the 10,000 simulations performed. Figure 2.3 shows the results. All of the values are close to 0 and the highest ones are close to the order of magnitude of the error given the number of simulations that were performed ($\frac{1}{\sqrt{10{,}000}}$).

In our implementations each card is controlled by a different processor. At the beginning of the execution each processor is assigned to a distinct GPU and it do not use any other one. The CUDA version used for this implementation did not allow the use of more than one GPU per processor, although more modern versions do. In any case it is faster to parallelize the host - device coordination than to use the serialized version currently available.

We generate 2 MPI process, one on each computer. Each of these span a number of light-weight threads equal to the number of GPUs in the machine using the pthread library. The communication between the 2 computers is done through

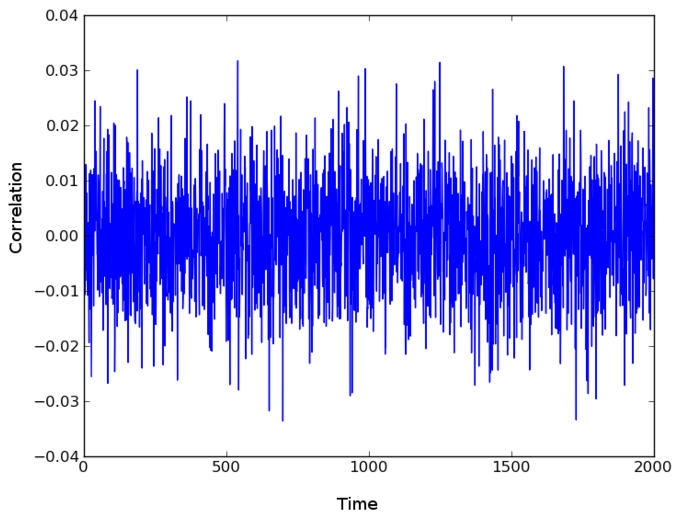


Figure 2.3: Correlation of random numbers used in a 1,000 neuron network simulation after 10,000 executions.

message passing while internally in each machine only shared memory is used. This allows us to reduce the use of the network, thereby increasing the execution speed. Another option would have been to create a different MPI process for each processor-GPU pair. In this case all the communication would have been done via message passing. Although modern MPI implementations benefit from shared memory when available there is a packaging process necessary for the creation of messages that our implementation avoids.

In order to numerically compute the righthand side of the system in the cluster, we first divide equally the number of neurons among the cards. For each time step the GPUs first compute the necessary random numbers using the Curand library. Then, one thread is created for each cell asigned to the card. Each of these compute the value for the next time step, for all the equations of a different neuron, using the Euler-Maruyama method. The threads are grouped in a one dimensional array, divided in blocks whose size maximize the usage of the processor (any factor of 32 which is the number of cores in one processor, see 1.3.4). Once all the threads in the GPU have finished, another function is run that sums the values of the all the y variables. Finally, the total is broadcasted by the use of shared memory.

Summing the values of an array in the GPU is not simple because the hardware is designed for SIMD operations, which means that each thread must be independent to obtain the best performance possible. One common way to compute a sum like this using multiple processes is to create a variable in shared memory and protect it from multiple accesses. This approach doesn't fulfill the constraints of GPU computing. We have solved this problem by creating a small number of threads and letting each of them compute a completely different partial sum. The final result can be obtained by adding the values computed at each thread. To enhance coalesced memory access each thread i iterates over the elements at positions (number of threads $\times j + i$) for $j = 0 \ldots \frac{\text{array size}}{\text{number of threads}}$.

Our approach only requires the transfer of one number at the beginning of each time step and of a small array at the end. This is much smaller than transfering the complete array for computing the sum at the CPU. Figure 2.4 presents a block diagram of the process. The computations performed in the GPU and in the CPU are divided into different zones and there are only 2 arrows that crosses from one to another, corresponding to the 2 memory transfers. Each of the squares in the GPU zone correspond to a different kernel, which are started by a call from the CPU once the previous has finished.

Figure 2.5 shows the correlation obtained for 3 different network sizes. The plot on the left shows the correlation for a 2 neuron network computed after 10,000 simulations, while the one in the center shows the same result but for a 1,000 neuron network. There is clear difference between the 2 as in the smaller network the correlation varies between -0.2 and 0.1 and in the bigger network it varies between -0.025 and 0.015. The results show that the correlation is indeed reduced when the amount of neurons is increased. The third plot in the figure, shows the same results as the one in the center but when the number of simulations is increased to 1,000,000. As the error in this case is smaller the correlations are reduced even more. We believe that the peak in the plot is due to the effect of the initial conditions, but even this high value is smaller that the ones obtained with less simulations.

Running 1,000,000 simulations to obtain the results in the plot at the right of figure 2.5 is a computational intensive task. For each simulation, at each of the 2,000 time step, the right hand side of 5,000 nonlinear equations need to be computed. This gives a total of 10^{13} function evaluations. For just one simulation the total number of uncorrelated random numbers generated is $2,000 \times 6 \times 1,000 = 12 \times 10^6$. The total amount of random numbers used in one complete execution is then 12×10^{12}. Dealing with this kind of figures is difficult for any personal computer using standard scientific computing software like Matlab or Python.

We have measured the amount of time taken for the simulation of 2,000 time steps for a 10,000 neuron network, with different amount of cards. In each case we average the time after 100 runs. The results are shown on the left of figure 2.6. The maximum speed is reached for three GPUs after that it starts to decrease. This is because after each computation of the right side of Equation (2.8), the threads that control each GPU must be coordinated, and the connectivity (y) needs to be shared before starting the next step. Synchronizing the threads is an expensive task that requires more time as more cards are used.

Another factor that could influence the change in speed when using more than three cards is the reduction in the number of threads inside each GPU. Indeed, when the number of neurons is kept constant, adding more cards implies that fewer neurons are assigned to each one; hence, fewer threads are used. Tesla cards are known to work better when there are many threads or the amount of computation in each thread is large enough. Without these conditions, the amount of time used in

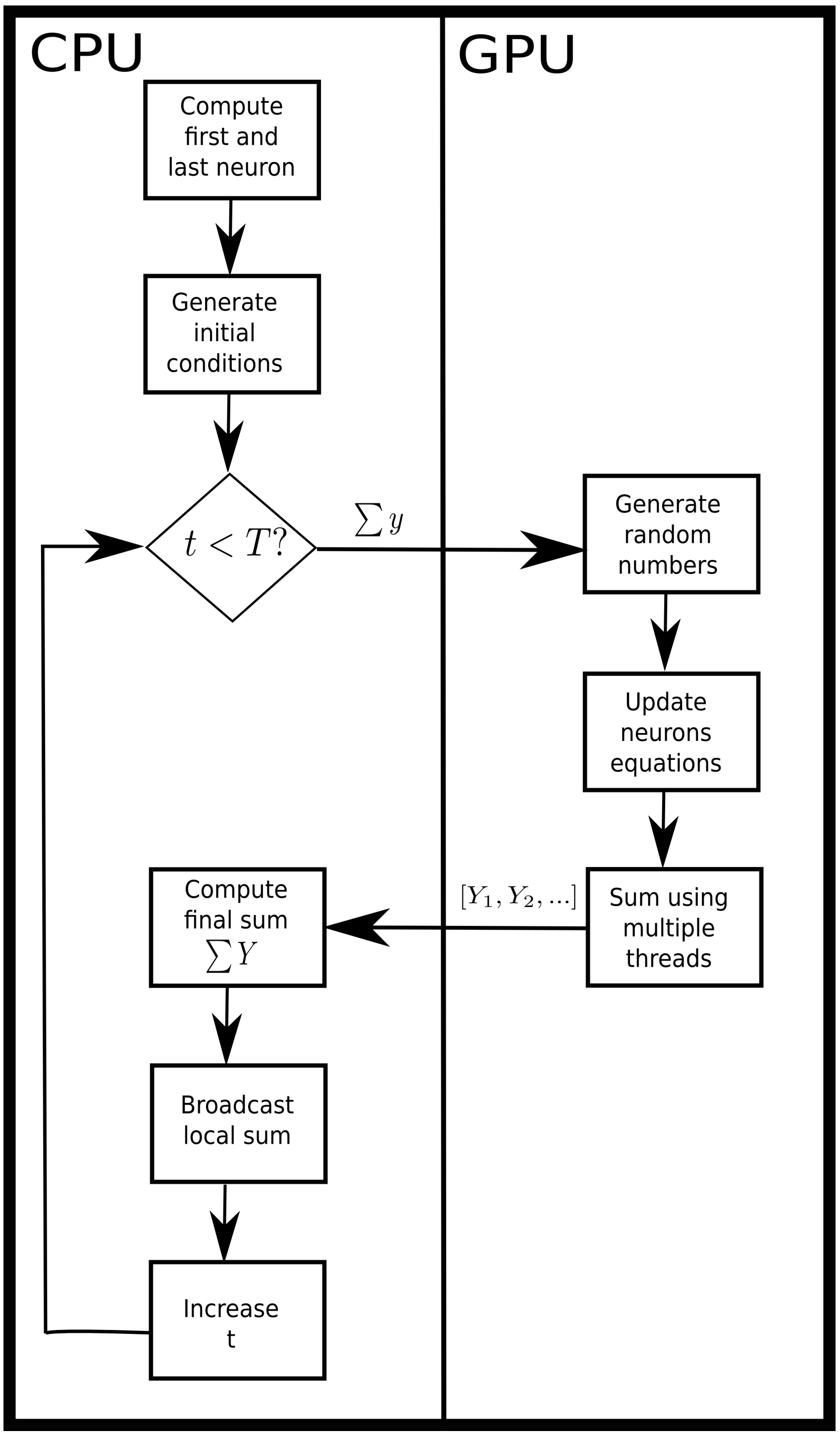


Figure 2.4: Flow diagram representing the Monte Carlo simulations of the Hodgkin-Huxley network. The values $Y_1, Y_2, ..,$ are the partial sums computed by each thread. See text for more details.

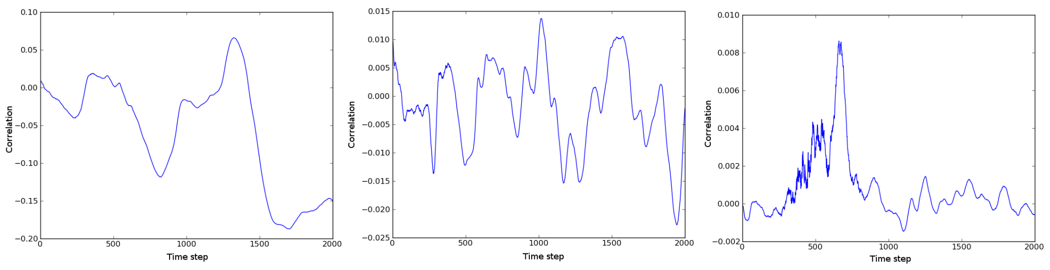


Figure 2.5: Left: time variation of the correlation in a 2 neuron network with 10,000 executions. Center: same with a 1,000 neuron network. Right: same as center but with 1,000,000 executions

memory transfer, context creation, and kernel launch inside the card will probably be too large compared to the real computation. To support this hypothesis, we increased the number of neurons to 100,000 and, as figure 2.6 shows, the maximum speed is reached for the six-card configuration: As more computations are needed for simulating the larger network, the total time is larger, but the largest speedup is reached for a larger number of cards than for a 10,000-neuron network.

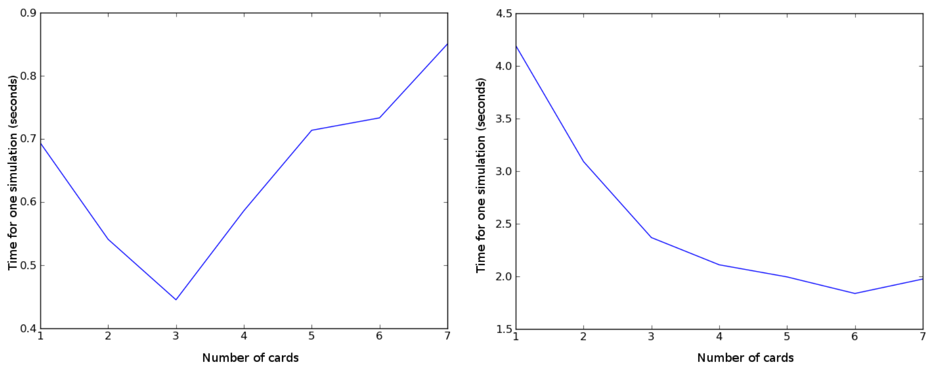


Figure 2.6: Left: time taken for one Monte Carlo simulation of 2,000 time steps for a 10,000 neuron network. Right: same for a 100,000 neurons network

2.2 One population Fokker-Planck equation

We have also solved the Fokker-Planck equation for the FitzHugh-Nagumo network using the GPU cluster available. In this section we describe the numerical methods and how they are implemented in the hardware. Then the result of several simulations are shown. As the Fokker-Planck equation describes the complete dynamics of a very large network, the results presented here show some interesting phenomena that can occur in extremely large systems.

2.2.1 Numerical method and implementation

We use the method of lines [Schiesser 1991] for solving the partial differential equation (2.11) when $P = 1$. We have chosen this method because of its simplicity and because it allow us to take the advantage of any approach designed originally for ODEs. It is also well suited for SIMD hardware architectures. The domain is discretized but the time is kept continuous. Each of the derivatives, with respect to the V, w and y variables are approximated using finite differences. This results in a big set of ordinary differential equations (one per point in the discretization), ODEs, that are solved using the GPU cluster.

First, we choose a finite volume in (V, y, w) space outside of which we assume that the probability density function is zero. This is done considering the maximum and minimum values seen for each variable in small network simulations. Then the volume is discretized in a total of $n_V \times n_w \times n_y$ points, where n_V is the number of points for the V variable, n_w for the w variable, and n_y for the y variable. The probability at each point outside the volume is considered to be 0 (Dirichlet boundary conditions).

The derivatives are approximated using the following fourth order central difference scheme, see [Khan 1999, Morton 2005]:

$$\frac{df(x)}{dx} \approx \frac{f(x-2\Delta x) - 8f(x-\Delta x) + 8f(x+\Delta x) - f(x+2\Delta x)}{12\Delta x}, \tag{2.13}$$

$$\frac{d^2 f(x)}{dx^2} \approx \frac{-f(x-2\Delta x) + 16f(x-\Delta x) - 30f(x) + 16f(x+\Delta x) - f(x+2\Delta x)}{12\Delta x^2}. \tag{2.14}$$

For the time integration the Runge-Kutta 4 scheme is used. The initial condition is given by the following Gaussian probability density:

$$p(0, V, w, y) = \frac{1}{(2\pi)^{3/2}\sigma_{V_0}\sigma_{w_0}\sigma_{y_0}} e^{-\frac{(V-\overline{V}_0)^2}{2\sigma_{V_0}^2} - \frac{(w-\overline{w}_0)^2}{2\sigma_{w_0}^2} - \frac{(y-\overline{y}_0)^2}{2\sigma_{y_0}^2}}. \tag{2.15}$$

With this approach the total number of ODEs we need to solve is $n_V n_w n_y$. This can become fairly large if we increase the precision of the phase space discretization. Moreover, increasing the precision of the simulation in the phase space requires, in order to ensure the numerical stability of the method of lines, to decrease the time step Δt used in the time integration scheme.

The Courant–Friedrichs–Lewy condition (see [Strikwerda 2004] for more details) states that in order for the method of lines to be stable with an explicit time integration the following relation must be fulfilled:

$$\Delta t \sum_{i=1}^{N} \frac{u_{x_i}}{\Delta x_i} \leq 1$$

where N is the number of dimensions, u_{x_i} is the derivative with respect to the variable x_i and Δx_i is the distance between 2 points in the x_i direction. This condition determines how much the time step must be reduced for finer discretizations.

It is possible to solve this equation with a very small grid and for short time periods in a personal computer using scientific software. The problem with this kind of approach is that due to the low number of points and large time step size, the instability of the method leads to the appearance of negative vealues. These are numerical errors that have an extremely strong effect on a probability density function which by definition can only have positive values. Our experiments also shows that the effect of the errors grows with time. This kind of small simulation can only be done for a small period of time after which errors dominate the solution and the shape of the probability density function is completely lost.

We have created the necessary software to solve equation (2.11) using the GPU cluster described in 2.1.4. The main objective was to study the effect of some parameters, mainly noise values, on the solution. For doing this we divide the computation into 2 steps: first the mean value of y (the integral in the right hand side) is computed and then the probability value at each point of the grid is updated. Being three dimensional, this integral is computationally demanding because all the grid points must be sampled.

The full domain is a cube, which is equally divided into smaller cubes, and in each combination processor/card the integral over this smaller domain is computed as follows. In each card, we first create a 2D array whose elements are the 1D integrals computed with respect to one of the three variables, keeping the other two constant. By summing the rows of the array, we obtain a 1D array of 2D integrals with respect to two of the three variables. These values are then sent to the CPU, where the final integral is computed and communicated to the other processes in the same machine via shared memory, and then to the others via an MPI message. Only one message is needed that contains the sum over all the integrals computed in the machine, having a small communication cost. Each processor after receiving the values from all the others, it adds them all up to obtain the whole integral.

The three steps required for computing the local integral in each processor-card pair are shown in the diagram of figure 2.7. Each dashed arrow in the diagram represents the direction of the integrals computed in each step. The process starts with a cube, then generates a matrix, then a vector and finally just one value. The arrows with solid lines represent the 2 kernels involved in the algorithm. Both of them work in a similar way, they create one thread per point of the new matrix/vector to be computed and then each solves one of the required integrals.

Each card computes the right hand side of equation (2.11) at all the points assigned to the card in the smaller cubes. One thread is created inside the card for each point, and it computes the corresponding value at that position. Because the sub domain is the same as it is in the integral, the data is sent only once to the GPU. Once the computation is finished the processor copies the boundaries of its assigned small domain to the shared memory. These values will be needed for the next computation in the neighboring cards. Finally the values at the half of the

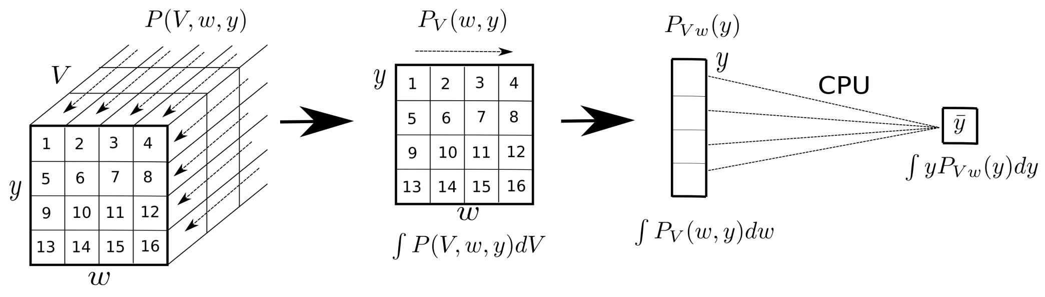


Figure 2.7: Diagram showing the process for the computation of $\bar{y}$

cube domain are sent via an MPI message. This is a bigger message than the one sent during the processing of the integral because it contains all the points assigned to one computer that will be needed by the others afterwards. The amount of values contained in this transmission is $4n_Vn_w$. The distribution of points is represented in the diagram of figure 2.8 and a flow chart of the process is shown in figure 2.9.

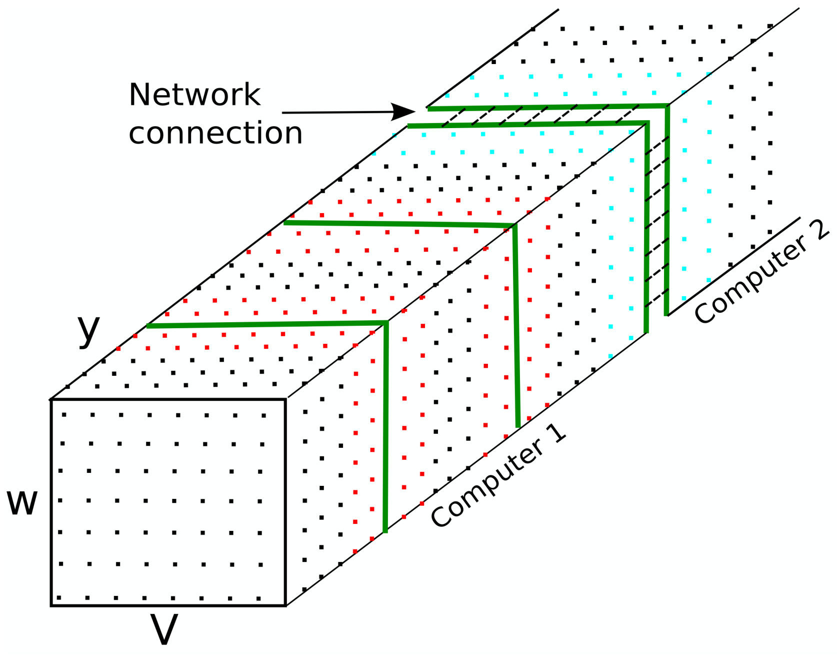


Figure 2.8: Diagram showing the organization of points in the 2 computers. The cube formed by all the smaller ones represent the complete V-w-y domain, which is cut in the y direction. The green lines indicate the limits between the processor-GPU pairs. The boundary points, in red, are transferred to shared memory after updating their value. The points in the limit between the 2 computers, in light blue, are sent through the network. The points in the middle of each sub-cube are kept in each GPU.

One different array is created in the GPU for the results of each of the 4 right hand calls necessary in the Runge-Kutta 4 scheme, one for each intermediate increment (k_1, k_2, k_3 and k_4). This is where the values are store after each call. The final values for the next time step are computed locally in each GPU by combining the 4 arrays following the integration method. Another set of 3 arrays is used for the input to each of the intermediate calls to the right hand side. The values of this arrays depends on the values of the probability at the last time step and on the previous right hand side call. One final array stores the solution obtained at the prior time. This gives a total memory usage of $\frac{8 n_V n_w n_y}{\text{num GPUs}}$.

A different approach for the computation of three dimensional finite differences is taken in [Micikevicius 2009]. At the time this work was performed, GPUs did not have any automatic cache capabilities and all the shared memory management had to be done by the programmer. All the techniques developed were designed to use the small amount of shared memory efficiently. Current cards, as the one used in this thesis, have an automatic cache and do not require this kind of method. This has also been mentioned in the study of [Michea 2010].

We have set the values of the weights in such a way that the probability of a synapse changing sign is 0. This is done by setting a synapse noise level ($\sigma_{\phi\gamma}$ in equation (2.6)) small in comparison to $\bar{J}_{\alpha\gamma}$, defined in equation (2.6). With this technique we avoid the use of the more complex connectivity of equation (2.7), which would increase the dimensionality of the resulting Fokker-Planck equation. These and other parameters that are common for all the simulations are presented in table 2.3.

The χ function used in all the simulations is:

$$\chi(y) = 0.1 e^{-0.5/(1-(2y-1)^2)}$$

Initial Condition	Phase space	Stochastic FN neuron	Synaptic Weights
$\overline{V}_0 = 0.0$	$V_{min} = -4$	$a = 0.7$	$\overline{J} = 1$
$\sigma_{V_0} = 0.2$	$V_{max} = 4$	$b = 0.8$	$\sigma_J = 0.01$
$\overline{w}_0 = -0.5$	$\Delta V = 0.027$	$c = 0.08$	
$\sigma_{w_0} = 0.2$	$w_{min} = -3$	$\sigma_w = 0.0007$	
$\overline{y}_0 = 0.3$	$w_{max} = 3$		
$\sigma_{y_0} = 0.05$	$\Delta w = 0.02$		
$\Delta t = 0.001$	$y_{min} = 0$		
	$y_{max} = 1$		
	$\Delta y = 0.003$		

Table 2.3: Common parameters used in all of the simulations of the Fokker-Planck equation

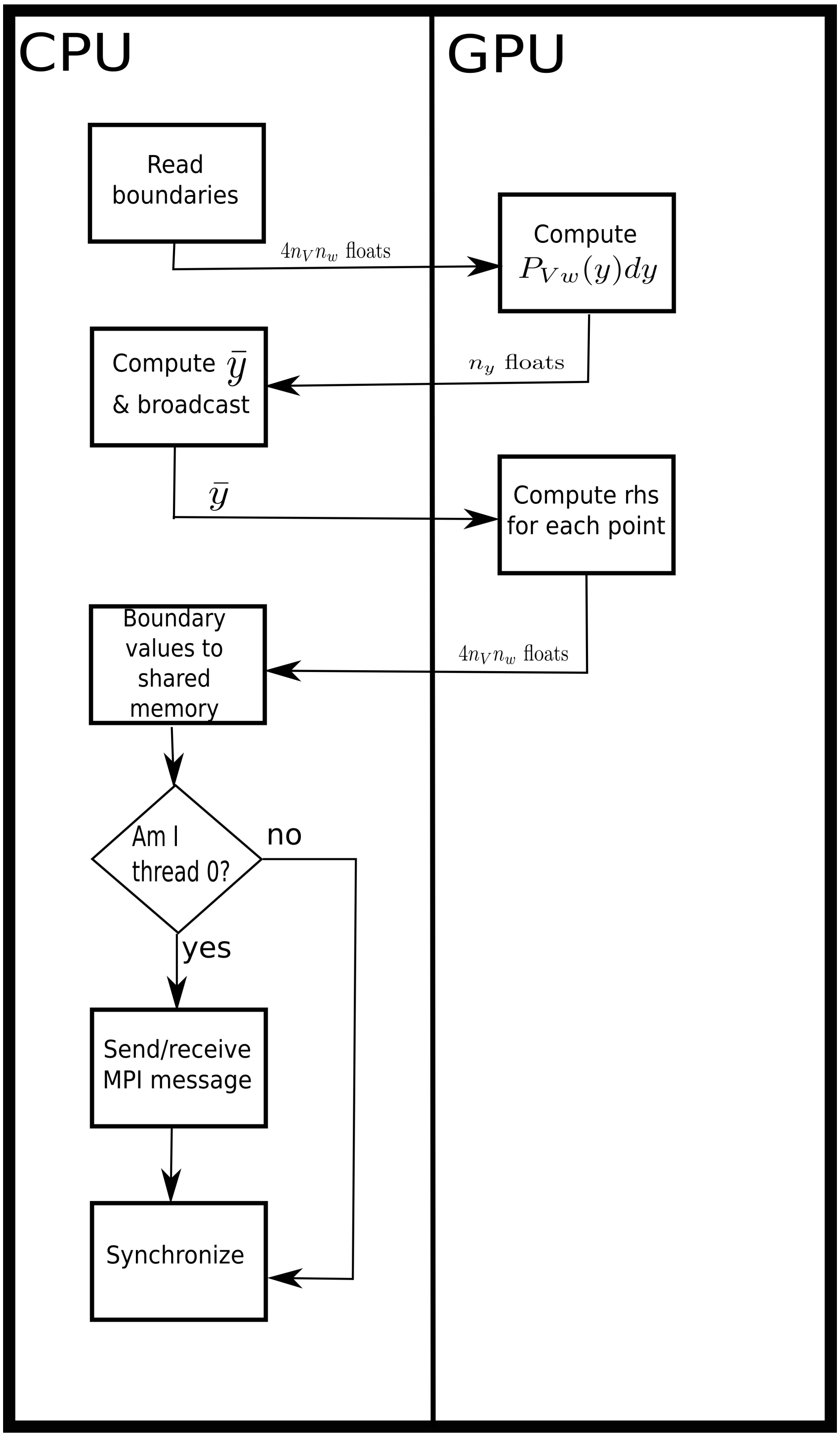


Figure 2.9: Flow diagram showing the different steps involved in each computation of the right hand side. The arrows that cross between CPU/GPU areas indicate how much data is sent.

A small independent noise, of intensity σ_w, is added to the w variable in order to make the numerical method stable. If the equation was left without noise the resulting Fokker-Planck equation would not include the second derivative with respect to the w variable, as in (2.11). Our numerical experiment have shown that the error provoked by the removal of any second derivative makes some probability mass go out of the defined volume, and then bounce back from the boundary. This effect creates negative ripples that should not be present in a probability distribution. The chosen value is small enough not to disturb the solution but keep the numerical method stable.

If the noise in the w variable needs to be removed a more complex finite difference approximation may be used. There are some schemes that are stable even if one of the drift terms is missing. One option is to use a Flux Limiting Scheme ([Roe 1986]), these are designed to limit the solution gradient near sharp changes in the solution domain. Also, an adaptive finite difference approximation, as the WINO scheme used in [Caceres 2011] for a Fokker-Planck equation obtained for a network of integrate and fire neurons could be implemented. In both cases it is necessary to study if the final equations are well suited for GPU computing.

2.2.2 Simulation results for the network of FitzHugh-Nagumo neurons

2.2.2.1 Stationary solutions

Four snapshots of the solution are shown in figure 2.10 (corresponding to the values $I = 0.4$ and $\sigma_{ext} = 0.27$) and three in figure 2.11 (corresponding to the values $I = 0.7$, $\sigma_{ext} = 0.45$). In the figures the left column corresponds to the values of the marginal $p(t, V, w)$, the right column to the values of the marginal $p(t, V, y)$. Both are necessary to get an idea of the shape of the full distribution $p(t, V, w, y)$. The first row of figure 2.10 shows the initial conditions. They are the same for the results shown in figure 2.11. The second, third and fourth rows of figure 2.10 show the time instants $t = 30.0$, $t = 50.0$ and the stationary solution. The three rows of figure 2.11 show the time instants t = 30.0, t = 50.0 and at convergence. In both cases the solution appears to converge to a stationary distribution whose mass is distributed over a "blurred" version of the limit cycle of the isolated neuron. The "blurriness" increases with the variance of the noise. The four movies for these two cases are available for download at http://www-sop.inria.fr/members/Javier.Baladron/thesis.html

The limit cycle of the isolated neuron is shown in figure 2.12. The plots there correspond to the result of simulations with just one unconnected neuron and the same parameters as in the experiment of figure 2.10 and 2.11. Each plot shows how the neuron traverses the phase space, going around a cycle, which has a similar shape to the one presented in the solution of the Fokker-Planck equation.

In order to characterize the solution of the Fokker-Plank equation we study how the dynamics changed when some parameters were varied. We focus our analysis on the different noise sources as it has been shown that they may not only have a disturbing effect but they may also improve the information processing capabilities of the brain [Rolls 2010]. Also on the work presented in [Touboul 2012] it is shown that the noise level may change completely the structure of the solutions of a similar equation coming from the mean field approximation of a firing rate model. For this reason, in the first set of experiments we made, the value of the external noise, σ_{ext} was changed, and all the other parameters were fixed.

2.2.2.2 Speed of convergence

In each experiment, with different external noise level, the convergence rate was measured using the following equation:

$$Cr = \int (P(t - \Delta t, V, w, y) - P(t, V, w, y))^2 dV dw dy.$$

To compute this quantity, first the difference between 2 consecutives steps is obtained, and then the integral is solved using the trapezoidal rule. If this value is close to 0 the solution is stationary while if it is large the solution may have big abrupt changes. The simulations have shown that the convergence is faster when the noise level is increased as can be seen on the plot of figure 2.13.

This plot also shows that the difference in convergence speed between noise levels is higher for low values. In fact, the curves for the 2 highest noise levels are very close to each other. This may indicate the existence of a limit in the amount of speed it is possible to gain by increasing the noise level. It is probable that for even higher levels of noise the stationary solution will not be achieved earlier in time.

2.2.2.3 Firing rate and mean voltage

The gain in convergence speed with higher noise levels is potentially useful only if the stationary solution found for each noise level is similar. If all solutions were completely different, noise would have only a disturbing effect and computing with higher levels of noise would require complex procedures to extract useful information. We have measured the mean voltage and the mean firing rate (to be explained below) in each case as shown in figure 2.14. In all cases the solution seems to be converging to the same values.

The mean voltage has a similar behavior but the bigger variability for lower levels of noise is more evident as can be seen in figure 2.14 top. The general tendency of all the simulations is to converge to a common value but the simulations with low levels of noise require more time. This can be noticed as the size of the waves is decreasing with time for all the curves

The firing rate was computed following the method proposed in [Fourcaud 2002] for integrate and fire neurons. There the Fokker-Planck equation is rewritten as:

$$\frac{\partial}{\partial t} p(t, \vec{X}) = -\vec{\nabla}_{\vec{X}} \vec{J}(t, \vec{X}), \tag{2.16}$$

where $\vec{X} = (V, w, y)$ and in our case:

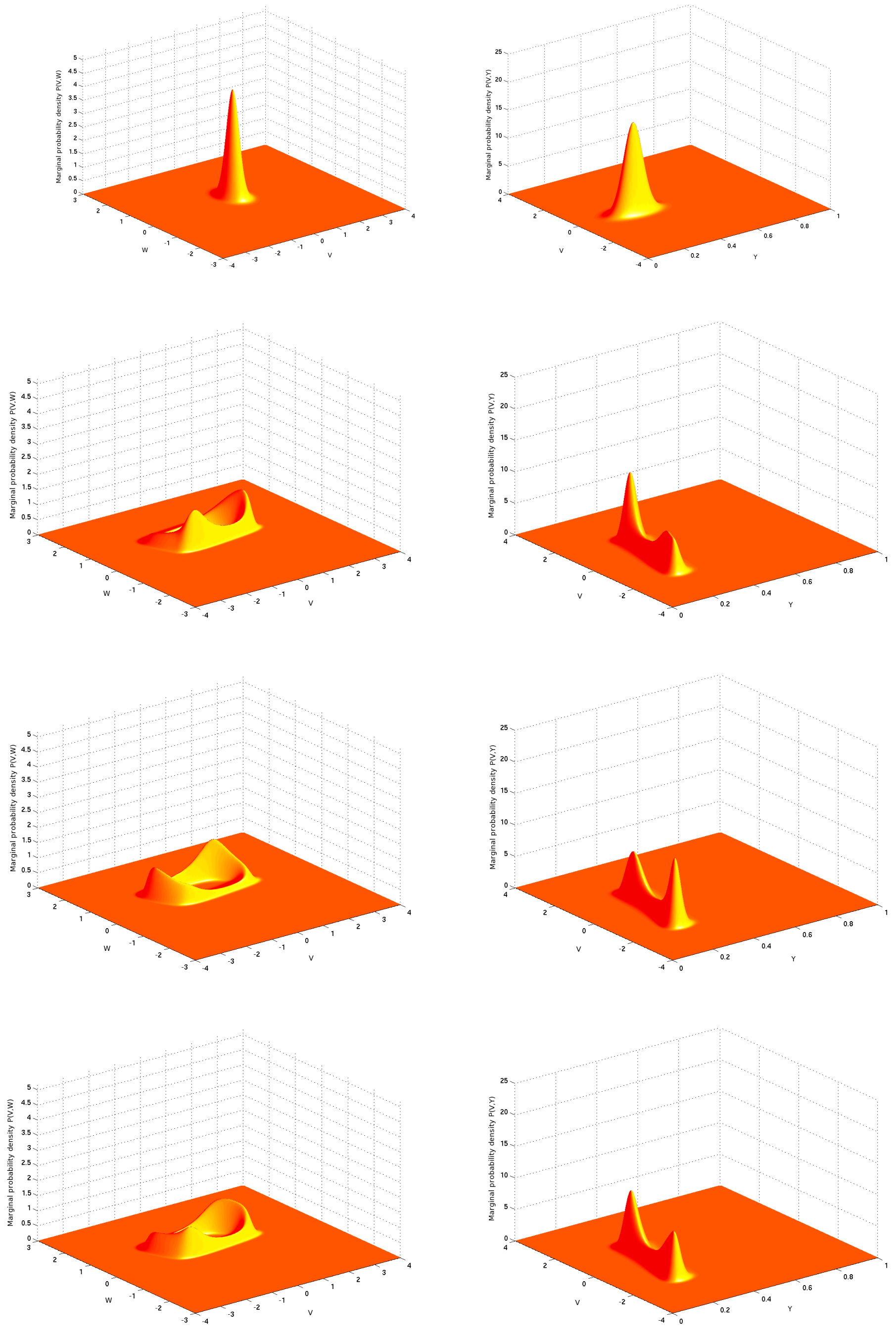


Figure 2.10: Marginals with respect to the V and w variables (Left) and to the V and y variables (Right) of the solution of the McKean-Vlasov-Fokker-Planck equation. The first row shows the initial condition, the second shows the marginals at time 30.0, the third the marginals at time 50.0 and the fourth the stationary (large time) solutions. The input current I is equal to 0.4 and $\sigma_{\text{ext}} = 0.27$. These are screenshots at different times of movies available in the web page.

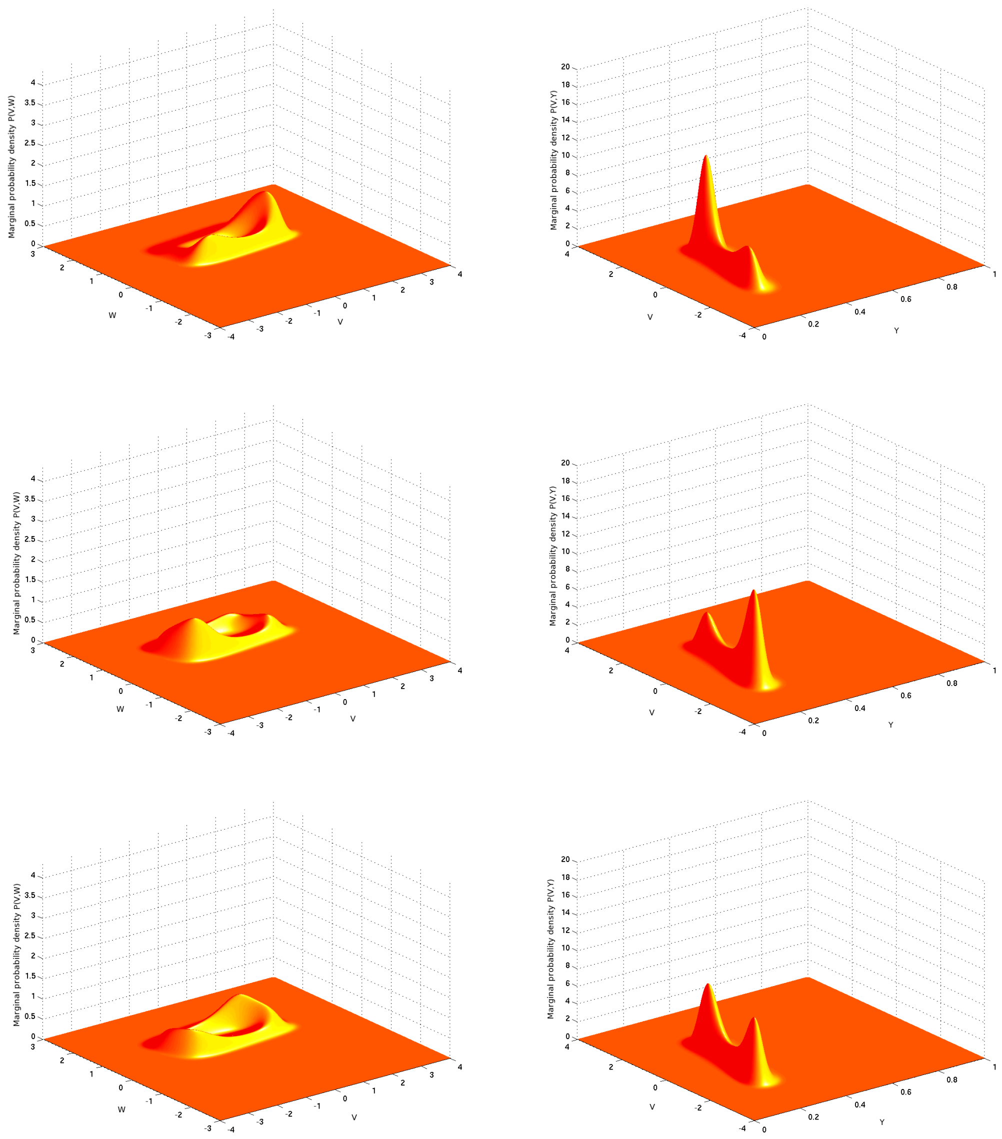


Figure 2.11: Marginals with respect to the V and w variables (Left) and to the V and y variables (Right) of the solution of the McKean-Vlasov-Fokker-Planck equation. The first row shows the marginals at time 30.0, the second the marginals at time 50.0 and the third the stationary (large time) solutions. The input current I is equal to 0.7 and $\sigma_{\text{ext}} = 0.45$. These are screenshots at different times of movies available in the web page.

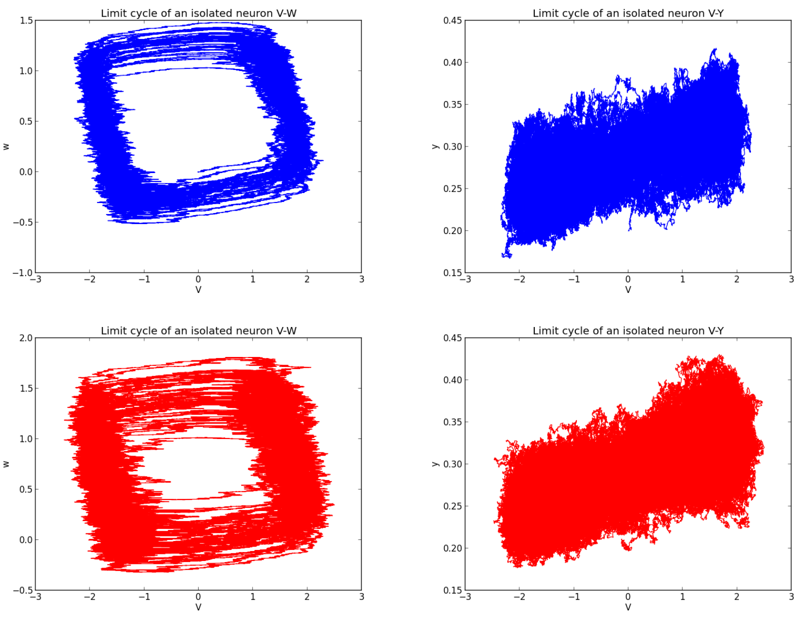


Figure 2.12: Traces of an isolated neuron showing the limit cycle. The first row corresponds to the a neuron with the same parameters as the experiment shown in figure 2.10 and the second to the one shown in figure 2.11. The first column corresponds to the V-w marginals and the second to the V-y marginals.

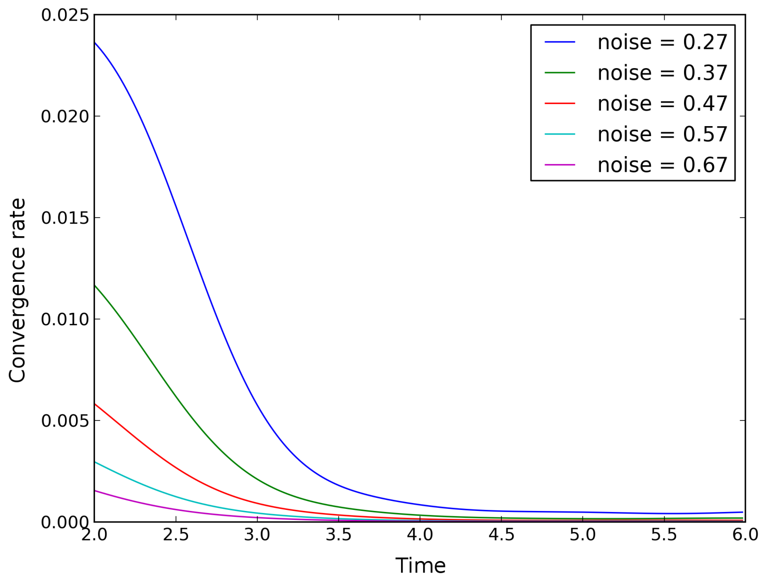


Figure 2.13: Convergence rate with different external noise levels for a limited period of time.

$$\vec{J} = J_V \hat{V} + J_w \hat{w} + J_y \hat{y}, \tag{2.17}$$

$$\begin{aligned}
J_V &= \Big[V - \frac{V^3}{3} - w + I - \bar{J}(V - V_{rev}) \int y' P(t, V', w', y') dV' dw' dy'\Big] P(t, V, w, y) \\
&\quad - \frac{1}{2}\frac{\partial}{\partial V}\Big\{\Big[\sigma_{ext}^2 + \sigma_J^2 (V - V_{rev})^2 \int y' P(t, V', w', y') dV' dw' dy'\Big] P(t, V, w, y)\Big\} \\
J_w &= c(V + a - bw) - \frac{1}{2}\sigma_w^2 \frac{\partial}{\partial w} P(t, V, w, y) \\
J_y &= [a_r S(V)(1 - y) - a_d y] P(t, V, w, y) - \frac{1}{2}\frac{\partial}{\partial y}\Big\{\Big[a_r S(V)(1 - y) + a_d y\Big]\chi^2(y) P(t, V, w, y)\Big\},
\end{aligned} \tag{2.18}$$

$\vec{J}$ is the probability current of the system, J_V , J_w , J_y are its components in the three directions of the phase space and finally $\hat{V}$, $\hat{w}$, $\hat{y}$ are the vectors that specify such directions.

Now, if we choose a volume Ω in the phase space with surface S_Ω and we integrate on this volume both the sides of equation (2.17) using the Divergence Theorem on the right hand side, we obtain:

$$\frac{\partial}{\partial t}\int_\Omega P(t, \vec{X}) d\vec{X} = -\oint_{S_\Omega} \vec{J}(t, \vec{X}) \cdot \hat{n} dS, \tag{2.19}$$

where $\hat{n}$ is the outward pointing vector normal to the surface S_Ω.

Therefore the right hand side of (2.19), without the minus sign, represents the amount of probability per unit of time that is flowing through the surface S_Ω. This value is positive if the probability is flowing outside the volume Ω.

Now, if we choose $\Omega = \{(V, w, y) : V \geq \Theta\}$, where Θ is a predefined threshold, this means that S_Ω is the hyperplane $V = \Theta$ and therefore $\oint_{S_\Omega} \vec{J}(t, \vec{X}) \cdot \hat{n} dS = \int_{\mathbb{R}\times[0,1]} \vec{J}(t, \Theta, w, y) \cdot \hat{V} dw dy = \int_{\mathbb{R}\times[0,1]} J_V(t, \Theta, w, y) dw dy$ represents the amount of probability that flows through it per unit of time.

This term contains the contributions of both the probability that is flowing outside (positive contribution) and the probability that is flowing inside (negative contribution) the volume Ω. More specifically, the positive contribution represents the case when the membrane potential is increasing in time and crosses the threshold $V = \Theta$ during the initial evolution of the spike, while the negative contribution represents the final part of its evolution, namely when the membrane potential is decreasing in time and crosses the threshold. These two contributions cancel each other when the solution is stationary, and therefore we want to keep only one of them in order to represent the spiking activity of the neuron

We can modify slightly the previous equation in order to fit it to our goal, which is to measure the amount of probability flowing through the threshold from below. In fact, we can see that:

$$\int_{\mathbb{R}\times[0,1]} J_V(t,\Theta,w,y)dwdy = \int_{\mathbb{R}} J_V^{marg}(t,\Theta,w)dw$$

where:

$$J_V^{marg}(t,\Theta,w) = \int_{[0,1]} J_V(t,\Theta,w,y)dy =$$

$$\Big[V - \frac{V^3}{3} - w + I - \bar{J}(V - V_{rev}) \int_{\mathbb{R}^2\times[0,1]} y'p(t,V',w',y')dV'dw'dy'\Big] p^{marg}(t,\Theta,w) \tag{2.20}$$

$$-\frac{1}{2}\frac{\partial}{\partial V}\Big\{\Big[\sigma_{ext}^2 + \sigma_J^2(V - V_{rev})^2 \int_{\mathbb{R}^2\times[0,1]} y'p(t,V',w',y')dV'dw'dy'\Big] p^{marg}(t,\Theta,w)\Big\} \tag{2.21}$$

Therefore the marginal probability current $J_V^{marg}(t,\Theta,w)$ is the $\hat{V}$ component of the probability current associated to the marginal probability density $p^{marg}(t,V,w) = \int_{[0,1]} p(t,V,w,y)dy$ evaluated on the hyperplane $V = \Theta$.

When the noise in the system is not small enough, the disturbance doesn't cause neurons to jump to a different branch of the limit cycle in the (V,w) plane. Each cell follows a trajectory close to the one of a noiseless neuron For this reason we have fixed our attention on it and not on the planes (V,y) or (w,y) which are shown in figures 2.10 and 2.11 left. Finally, we can isolate only the positive contributions to $J_V^{marg}(t,\Theta,w)$ and compute:

$$\int_{\Xi(t)} J_V^{marg}(t,\Theta,w)dw, \tag{2.22}$$

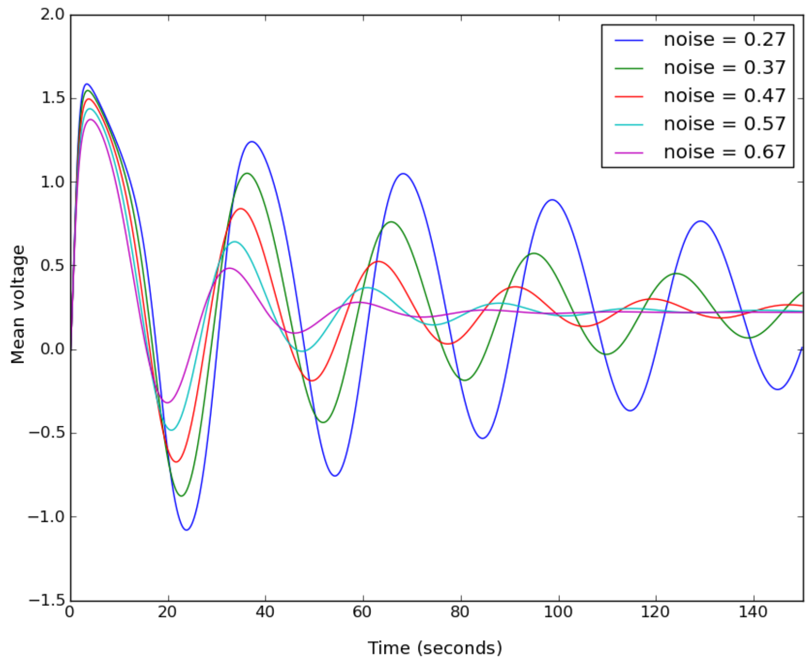
where $\Xi(t) = w : J_V^{marg}(t,\Theta,w) > 0$ on the line $V = \Theta$

If we interpret $p(t,V,w,y)dVdwdy$ as the probability of finding the state of a given neuron in the cube $[V,V+dV]\times[w,w+dw]\times[y,y+dy]$ at time t (this is possible due to the independence resulting from the propagation of chaos effect), equation (2.22) is the mean firing rate of a real neuron.

As shown in figure 2.14 the above quantity is similar in all the simulations. The difference between them can be found when computing the variance. As can be expected, higher levels of noise produce a higher variance. Noise makes neurons deviate from the limit cycle before being reatracted to it. This may cause some neurons to move slower or faster than a noiseless cell, as they do not follow the cycle exactly. Because noise is different for each element they end up spreading over all the possible values. Figure 2.15 shows the variance in all the cases.

2.2.2.4 Effects of the deterministic input value

In the next set of experiments we fix the external noise but changed the input value. For every case we compute the mean firing rate as described above. This should give us an idea of the information processing capabilities of the network.



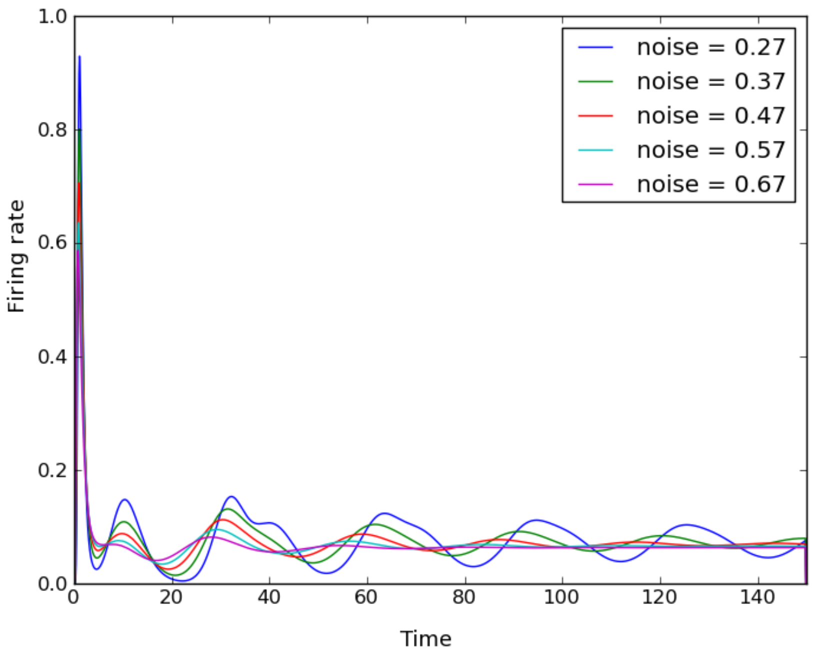


Figure 2.14: Top: mean voltage for the different external noise levels. Bottom: mean firing rate for the different external noise levels

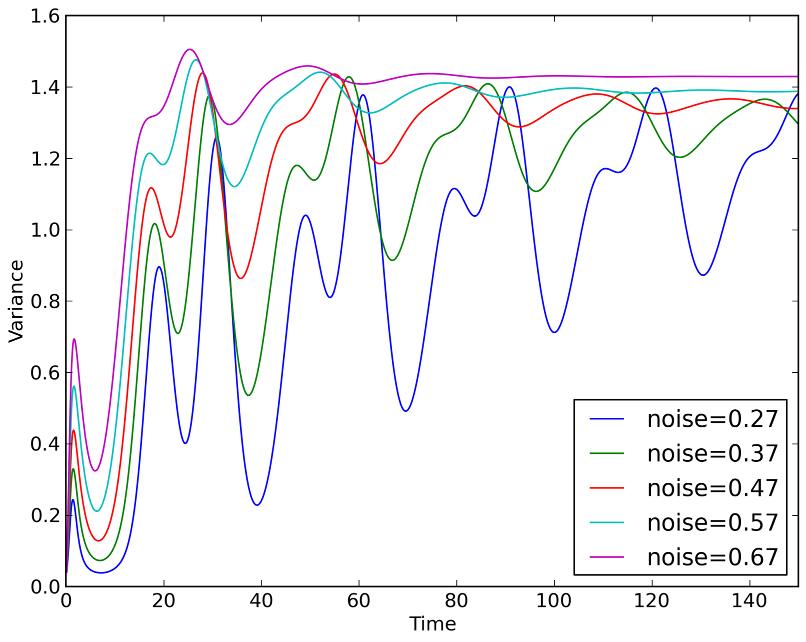


Figure 2.15: Variance for simulations with different external noise levels

Figure 2.16 shows the firing rate as a function of the input value for the last time step of our simulation. The images there are just a screenshot of 2 movies available at the web page. The plots and movies show how the firing rate increases with the input until it reaches a maximum value and then starts falling. This happens for the 2 levels of noise and can be seen during the whole simulation. The reason for this drop is that at the input value that produces the maximum firing rate the limit cycle for the isolated neuron has disappeared. The solution to the mean field equation for high input values is just a single bump whose center position depends on the input value. Increasing the input moves the center further from the spike threshold producing the decrease in the firing rate. Larger levels of noise increase the width of the bump.

In order to further check this we have solved the mean field equations (2.10) with a high input value that should not produce any spiking. We have taken the probability distribution computed by solving the Fokker-Planck equation and use this together with the mean field equation to generate samples of the process. Some samples are presented in figure 2.17. The examples show that the voltages fluctuate around a mean value, which is close to 1.5. No clear spikes can be detected in the shape of the curves, unlike in figure 2.1.

Both movies show that the firing rate converges quickly for high levels of input while it continues to oscillate during the whole duration for very small values. We computed the convergence rate for some input values where the limit cycle exists and

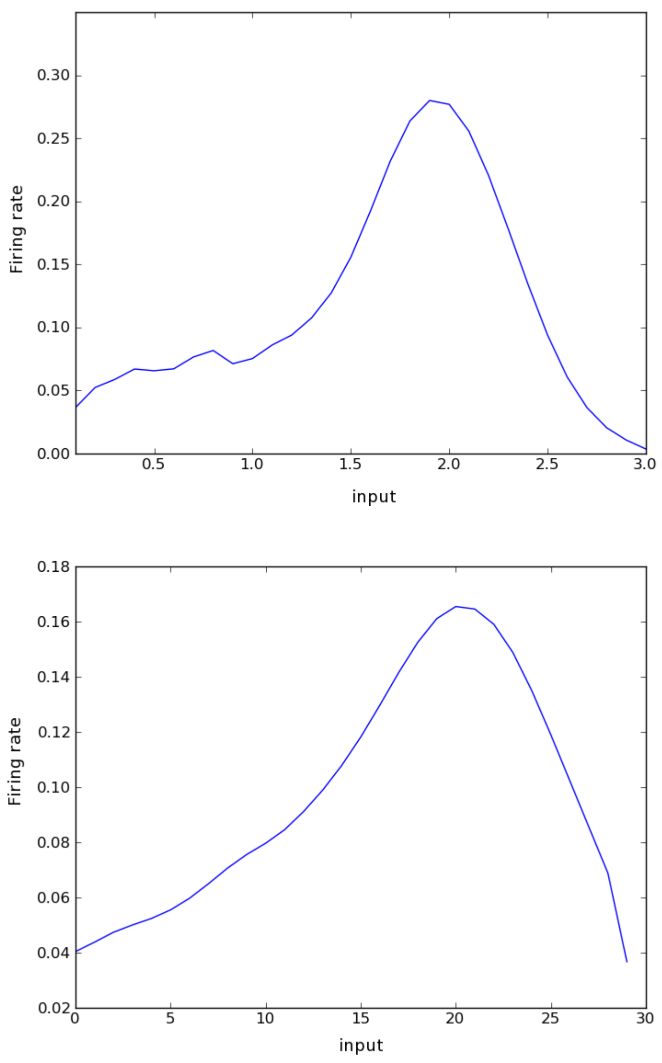


Figure 2.16: Firing rate for different input value at the final time step of the simulation. The external noise is 0.27 for the plot on the top and 0.45 for the one on the bottom. Movies for both simulations can be found on the web page.

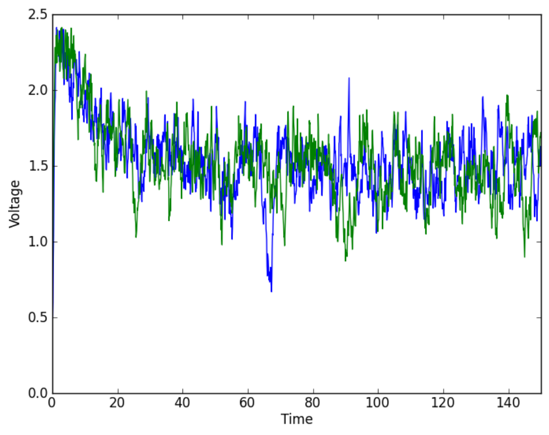
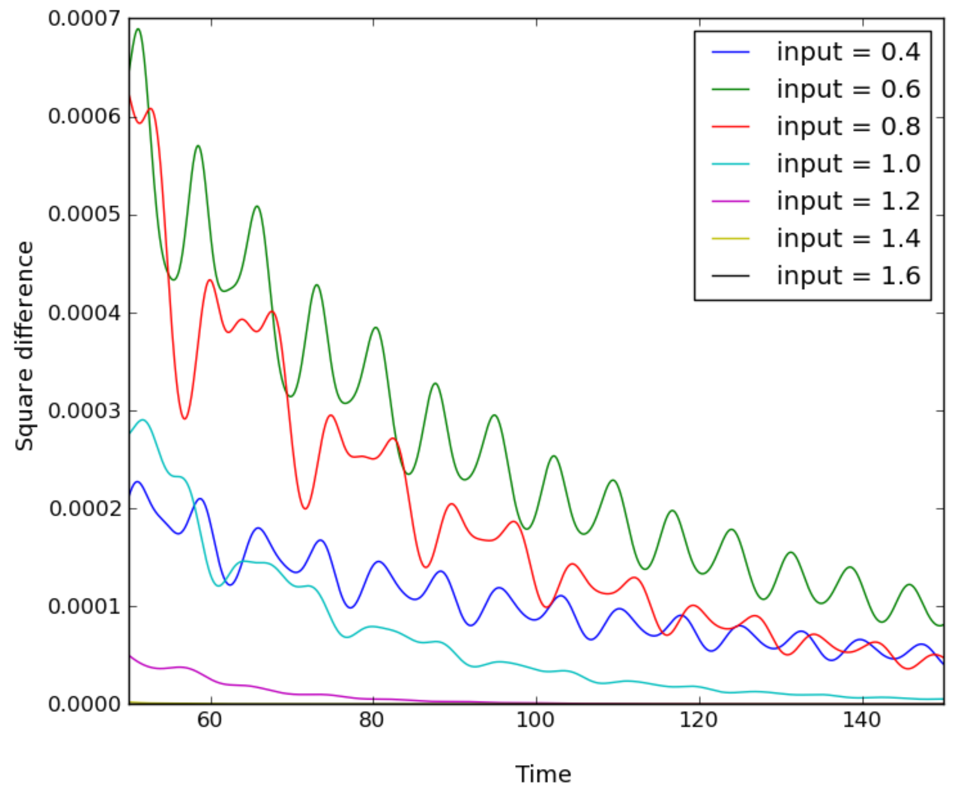


Figure 2.17: The green and blue line are voltage traces obtained as samples from the stochastic process described by the mean field equation. The input value used for the simulations is high, in a level where no spiking occurs

set the external noise level to 0.27 and 0.45. These results are shown in figure 2.18 where a plot of the convergence for the final time steps of the simulation is presented. In both cases, starting from input 0.6, the convergence is faster with higher input. This phenomenon is similar to what was seen for the external noise levels. In the case of input 0.4 the convergence is faster than other higher values of noise, not fulfilling the previous pattern. This is because the value of the deterministic part of the input is too low compared to the noise level. In this case the input signal the neurons are receiving is mainly composed of noise

In a third group of experiments the value of the noise level at the synapses, σ_J, was varied and the rest of the parameters were fixed. This is the second important source of noise of the model, and as the first one (σ_{ext}), has important effects on the solution we expected this one to completely change the final probability density. This was not the case, as with the 3 configuration we tested, presented in figure 2.19, no important change, neither in shape nor in convergence, was detected

In the final experiment with the Fokker-Planck equation, we used an input function that changed in time. We let the system converge to the stationary regime and then change the input value to a slightly smaller one. This allow us to see the changes in the probability distribution when the initial conditions are distributed along the limit cycle and determine how fast the system can react to fluctuations in the input. The firing rate for this experiment is presented in figure 2.20 and the movie for this simulation is included in the web page. The results show that the system was able to react very fast, as the probability start to change as soon as the input varies. The convergence to a new stationary solution is achieved faster that with the initial condition from the previous experiments.



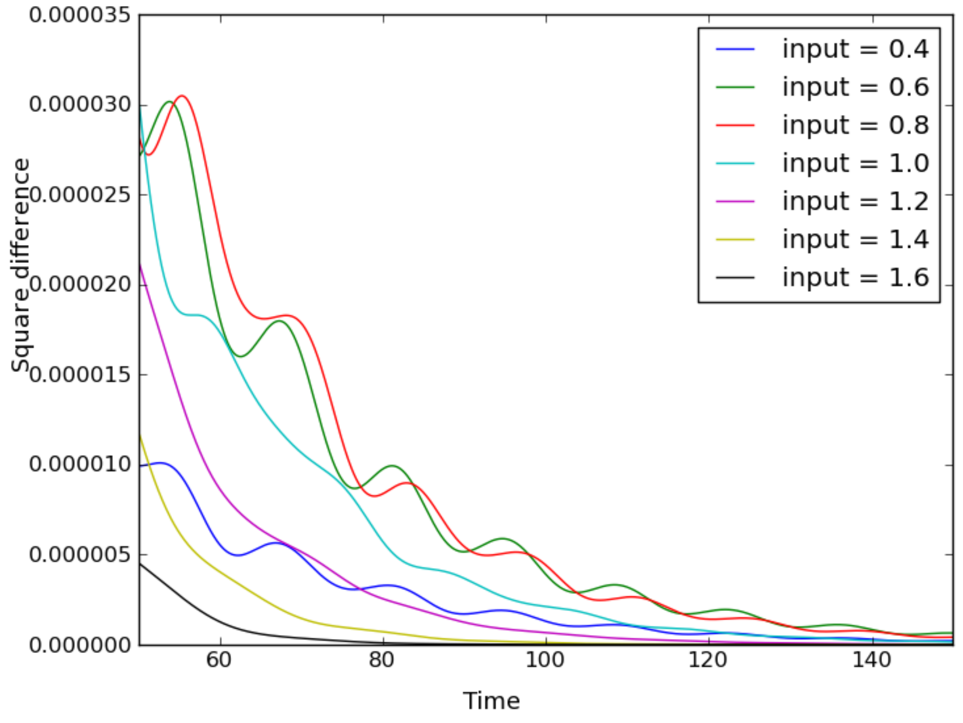


Figure 2.18: Convergence rate for the last time steps of the simulation. The external noise for the plot on the top is 0.27 and for the one on the bottom is 0.45.

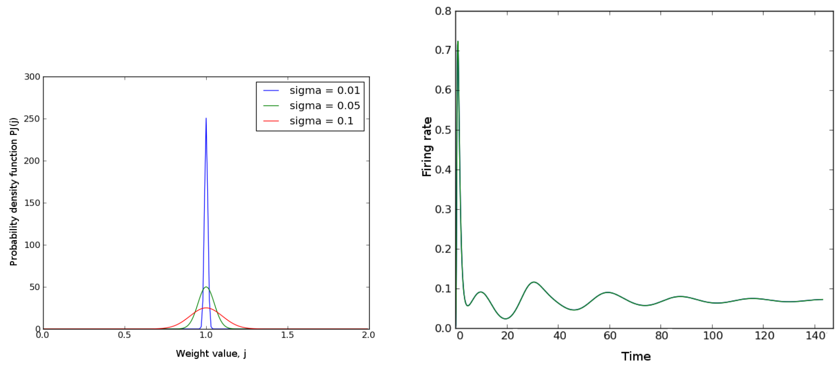


Figure 2.19: Left: probability distribution for the different diffusion processes used in the experiments. This describe the law of the possible values taken by the weights. The max conductance of the process is 1 in both cases. The width of the curve is given by the value of σ_J. Right: the firing rate obtained with the 3 different synapse configurations

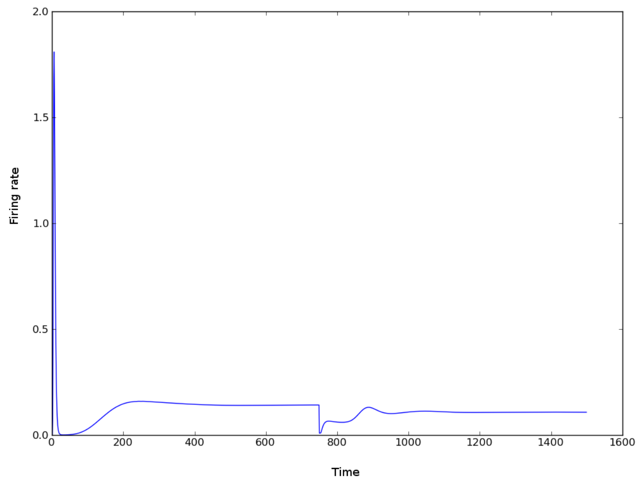


Figure 2.20: Firing rate for a simulation when the external input value is varied at time step 750. The plot shows how the system adapts to this new input value. A movie of this simulation can be found in the web page

2.2.2.5 Comparison to isolated neuron

To determine the effect of the network on the dynamic, we computed the probability density function of an isolated neuron with the same parameters as in the experiments presented in figure 2.10 and 2.11. To do this we used Monte-Carlo simulations in a similar way as was done for the Hodgkin-Huxley network in section 2.1.4. In each simulation we considered a sample of the process and used this to construct an approximation of the probability density function. We have chosen this method, instead of solving the corresponding Fokker-Planck equation, because it requires solving just 3 stochastic equations instead of a large set of ODEs. Solving the Fokker-Planck equation would have required the same discretization size as before (to be able to compare) and would take a similar amount of computational time. Instead, solving directly the neuron equation allow us to create a faster implementation, especially if we use a parallel random number generator like Curand.

The simulation with external noise 0.27 and input value 0.4 of an isolated neuron didn't converge to a stationary solution as the one showed in figure 2.10. A movie for this simulation is available at the web page and a screenshot of the final time step is presented in figure 2.21 left. Instead, the experiment with noise 0.45 and input 0.7 did converge but to a different solution than the one of the network (shown in figure 2.11). A movie for this experiment is also available at the web page and a plot of the stationary solution is shown in figure 2.21 right. The quality of both movies is worse than the experiments with the Fokker-Planck equation because we have only used 10,000 simulations. Increasing enough this number should produce a similar quality but would slow down the simulations

The effect shown by these simulations is similar to what was shown on those with different input values (see figure 2.18). The input to each neuron is composed of the external signal plus the current sent through synapses from other cells. The total input of the isolated neuron is smaller than the one a cell in a network receives (there are only excitatory connections). Given the behavior found in previous simulations, the system should converge only if the input is high enough, which may not be the case in the first isolated neuron experiment but may happen in the second. This is an example of an interesting change in the dynamics, produced by letting a neuron interact with others.

2.2.3 Simulation results for the network of Morris-Lecar neurons

We repeated some of the previous numerical experiments with the more realistic Morris-Lecar neuron model described by equations (1.6). The shape of the spikes produced by this model is closer to those produced by the Hodgkin-Huxley type than the ones produced by the FitzHugh-Nagumo model. Also the parameters in the model can be related to physical quantities, providing units for them and allowing to compare the results with the real measurements.

A similar approach to the one used to include noise in the FitzHugh-Nagumo model was used to create a network of noisy Morris-Lecar cells. First the input

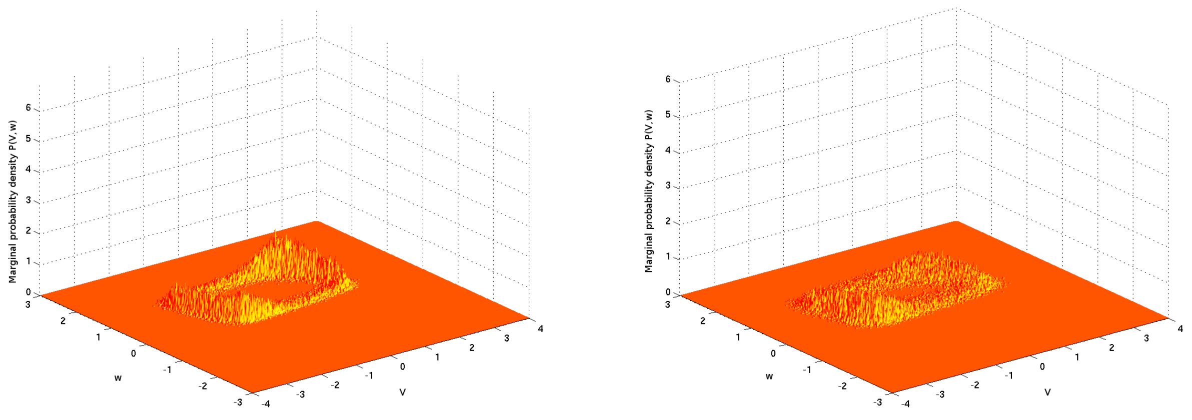


Figure 2.21: Probability density for 1 neuron approximated after 10,000 Monte Carlo simulations. Left for experiment with noise 0.27 and right for noise 0.45.

function was separated in a stochastic and a deterministic part. This allows us to determine if the effects of the external noise are the same in this model as in the previous one. Another small noise source was included in the recovery variable for the same reasons as before. No change was made in the synapse structure. The common parameters for all the simulations are presented in table 2.4. The resulting Fokker-Planck equation is:

$$
\begin{aligned}
\partial_t P_\phi(t,V,w,y) = \sum_{\gamma}^{P} \frac{1}{2}(\sigma^J_{\alpha\gamma})^2 \bar{y}^2_\gamma(t) \frac{\partial^2}{\partial V^2}\left[(V - Vrev)^2 P_\phi(t,V,w,y)\right] + \\
\frac{1}{2}\frac{\partial^2}{\partial y^2}[\sigma^2_Y(V,y)P_\phi(t,V,w,y)] + \frac{1}{2}\sigma^2_{ext}\frac{\partial^2}{\partial V^2}[P_\phi(t,V,w,y)] \\
- \frac{\partial}{\partial V}\Big[((-g_{Ca}Mss(V)(V - E_{Ca}) - g_K W(V - E_w) - g_L(V - E_L) + I)\frac{1}{C} \\
-\sum_{\gamma}^{P} \bar{J}_{\alpha\gamma}(V - Vrev)\bar{y}_\gamma(t))P_\phi(t,V,w,y)\Big] - \frac{\partial}{\partial y}\left[(\alpha_r S(V)(1-y) - \alpha_d y)P_\phi(t,V,w,y)\right] \\
- \frac{\partial}{\partial w}\left[((W_{ss}(V) - w)/T_W(V))P_\phi(t,V,w,y)\right], \quad (2.23)
\end{aligned}
$$

The simulations with this model show the existence of stationary solutions similar to the ones found for the FitzHugh-Nagumo model. An example of a stationary solution can be seen in figure 2.22. The (V,w) marginal probability density spreads around the limit cycle in a similar fashion to the FitzHugh-Nagumo model. The structure of the 2 limit cycles whichx] produce the spikes are similar. Again the "blurriness“ depends on the noise values.

Initial Condition	Phase space	Stochastic ML neuron	Synaptic Weights
$\overline{V}_0 = -10.0$	$V_{min} = -60$	$g_{Ca} = 4.0(mmho/cm^2)$	$\overline{J} = 0.3$
$\sigma_{V_0} = 4.0$	$V_{max} = 60$	$g_K = 8.0(mmho/cm^2)$	$\sigma_J = 0.01$
$\overline{w}_0 = -0.2$	$\Delta V = 0.38$	$g_L = 2.0(mmho/cm^2)$	
$\sigma_{w_0} = 0.05$	$w_{min} = -0.2$	$E_L = -60.0mV$	
$\overline{y}_0 = 0.2$	$w_{max} = 0.6$	$E_K = -84.0mV$	
$\sigma_{y_0} = 0.05$	$\Delta w = 0.003$	$E_{Ca} = 120.0mV$	
$\Delta t = 0.001$	$y_{min} = 0$	$V1 = -1.2mV$	
	$y_{max} = 0.8$	$V2 = 18.0mV$	
	$\Delta y = 0.003$	$V3 = 12.0mV$	
		$V4 = 17.4mV$	
		$C = 20$	
		$\sigma_w = 0.01$	

Table 2.4: Common parameters used in all of the simulations of the Fokker-Planck equation for the Morris-Lecar model

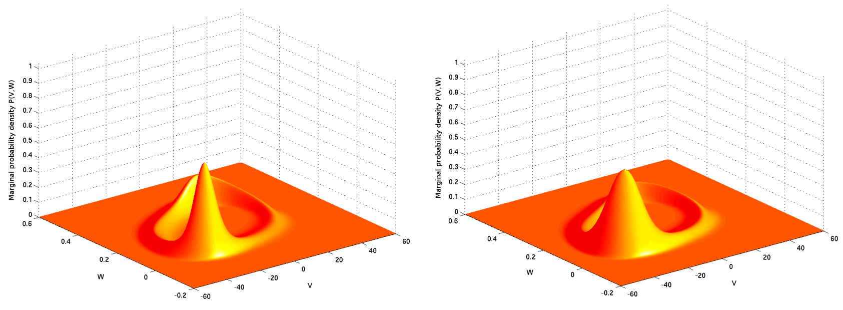


Figure 2.22: Example solution of the Fokker-Planck equation for the Morris-Lecar model. In the left the solution at time 50 and in the right the stationary state. Movies for this simulation are available on the web page

The solutions were obtained using a modified version of the same code. The job distribution and communications scheme were left the same, only the equation was changed. As the range of values for the different variables in the Morris-Lecar model is very different from those in the Fitzhugh Nagumo model the limits of the volume are different.

The distribution of points in the grid had to be changed in order to keep a similar error level in the estimation of the derivatives by finite difference. As the voltage varies between -60 and 60 in the Morris-Lecar model, while it varies between -4 and 4 in the FitzHugh-Nagumo, more points are necessary in the discretization for V to keep a ΔV similar to the previous experiments. To do this without increasing the computational complexity too much, the range of values for the y and w variables were reduced and discretization points were taken out of these dimensions and added in V. Finally, the total amount of points is similar in both cases, but the points are distributed differently.

When high input values are presented to the Morris-Lecar model the limit cycle is destroyed and only a stable fixed point is present (see bifurcation diagram in [Izhikevich 2007]). This is the same effect that made the firing rate of the FitzHugh-Nagumo network decrease after the peak in figure 2.16. Figure 2.23 shows the final stationary probability density in one of these cases. All the mass is distributed around the stable fixed point and the width of this peak depends on the noise levels.

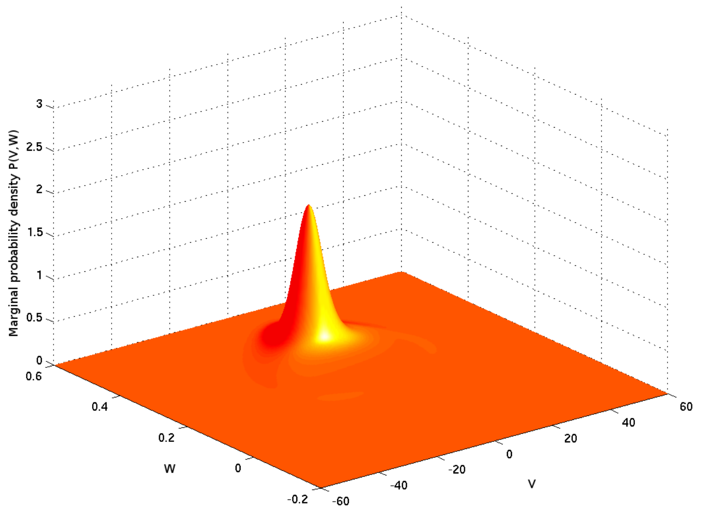


Figure 2.23: Stationary solution for the Morris-Lecar model with high input

We have repeated the experiments reported in section 2.2.2 where we varied the external noise with the Morris-Lecar model. As in the previous case, the convergence rate was measured each time. A similar behavior was found, the highest the noise levels, the faster the convergence. The limit for the gain in speed due to the rise in external noise also seems to exist in this case. Due to the difference in the range of possible values for the voltage variable the external noise used for these experiments is much higher than before. See figure 2.24 for details on the different convergence rates.

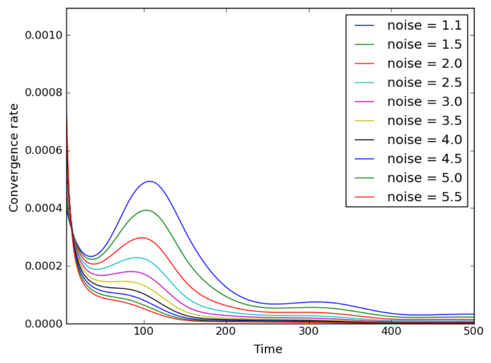


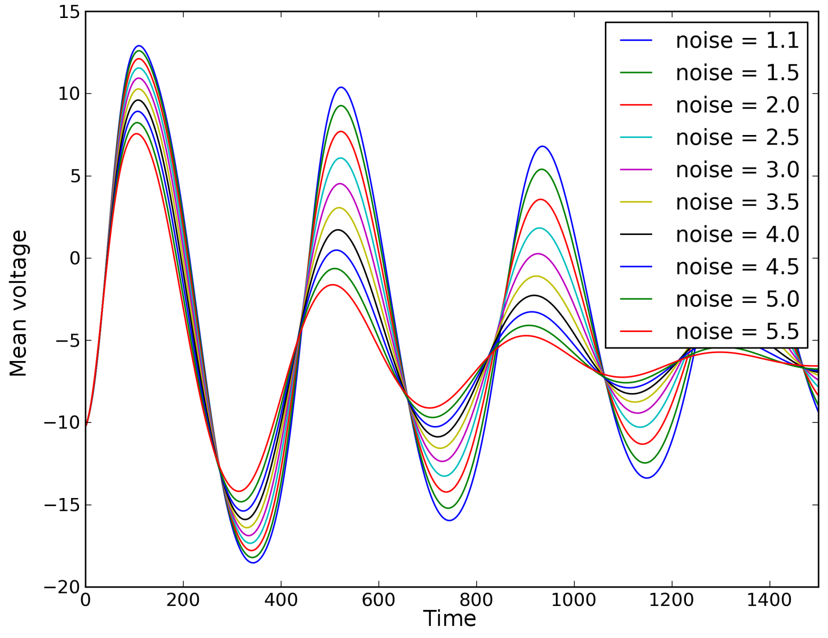
Figure 2.24: Convergence rate for several simulations of the Morris-Lecar network with different external noise levels and input = 80.

The same procedure as in the FitzHugh-Nagumo case was used to characterize the solution for each noise level. The mean voltage value and the mean firing rate were computed and compared. The results can be seen in figure 2.25. The mean voltage has a similar behavior to the previous experiments, for all noise levels it converges to a similar value. The plot also shows how the higher levels of noise are approaching this value faster. The firing rate behavior is different from before, in this set of experiments it is reduced by a small amount when the noise level is increased. This difference is small and although the solutions are not the same they are similar enough for the system to benefit from the faster convergence. Finally, the variance was also computed for these simulations (see figure 2.26), obtaining a structure similar to that of the FitzHugh-Nagumo case.

A somewhat surprising behavior was found in simulations with small inputs and high levels of noise. The firing rate in these cases doesn't converge but it decreases over time. Although this decrease is slow, it looks like the value of the input is not high enough to dominate the behavior of the neuron as it does when the noise is smaller. Figure 2.27 shows the mean firing rate for 3 different input values and noise intensity 5.5, the plot shows how for input 60, 80 and 100 the system converges to a stationary solution while for input 40 it is decreasing. The figure also includes the rates for input 40 and 80 when the noise level is 3.5. In this case the firing rate converges for the small input. This effect was not present in the FitzHugh-Nagumo simulations.

2.2.4 Speed of our implementation

Obtaining the previous results is a computationally demanding task mainly due to the number of points in the grid, the nonlinearities in equation (2.11) and the integral term. GPU computing techniques allowed us to solve the large system of



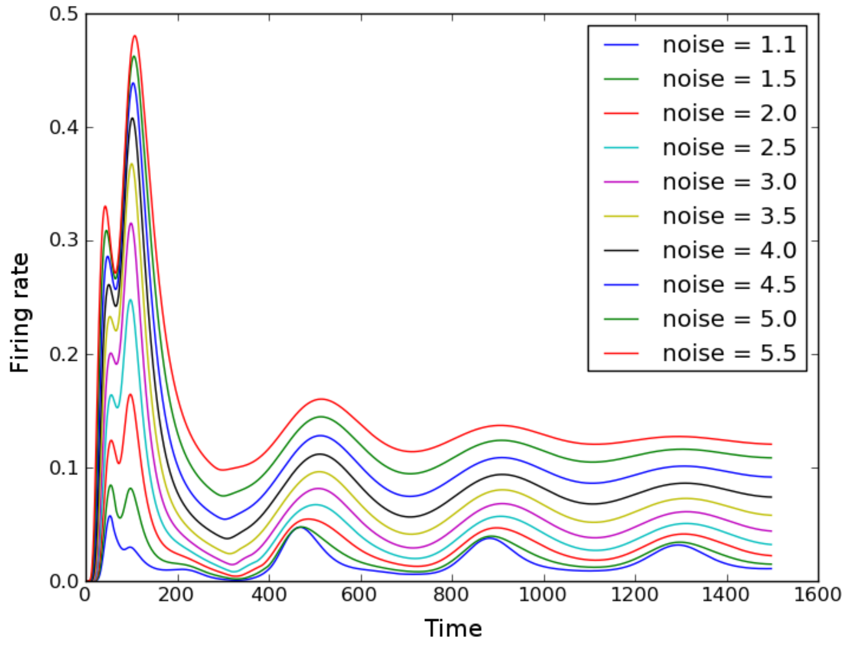


Figure 2.25: Top: mean voltage. Bottom: mean firing rate.

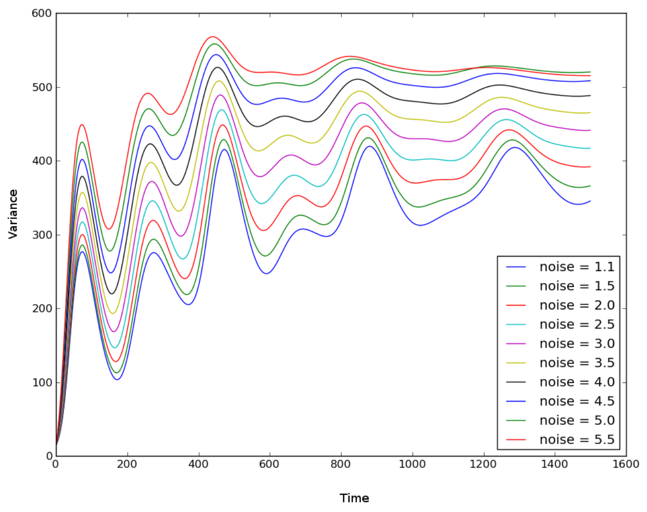


Figure 2.26: Variance for simulations of the Morris-Lecar network with different noise levels.

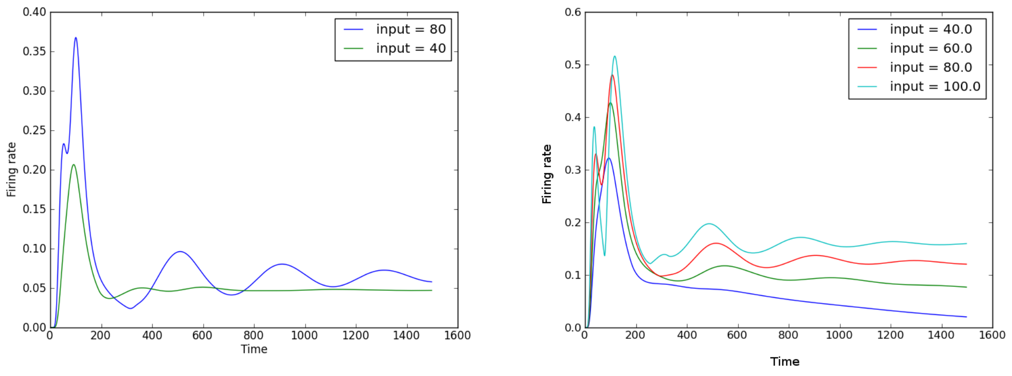


Figure 2.27: Mean firing rates for different input values with 2 different levels of external noise. The plot on the left shows the rates with noise level 3.5 and the one on the right with noise level 5.5.

non linear differential equations fast enough to be able to study the dynamics of the system when several important parameters change. The evolution of the solution for long periods of time is hard to obtain due to the stiffness of the equations which forbids us to use large time steps. In fact, before the use of a parallel machine, we could only compute the dynamics for a small period of time and with large error bounds.

For this kind of application GPUs are a competitor for large scale standard clusters. The mean execution time for one time step of the FitzHugh-Nagumo Fokker-Planck equation (measured after 100 repetitions) is **0.06 seconds** when the 2 computers and the 14 cards are used. This includes computing the righthand side of the system of ODEs created by discretizing equation (2.11) 4 times and then combining these results to get the final value for the current time. In order to reach the same speedup without GPUs we would require a machine with several hundreds processors, which is much more expensive and difficult to maintain and use than our hardware.

A set of experiments was designed to compare our approach with other possible solutions. First, the code was changed in such a way that all the computations that were done in the GPU were executed in the processor itself. The amount of points assigned to each processor is the same as before but this time there is no GPU to update the values in parallel. With this approach one level of parallelism is removed by sequentially computing the right hand side for each point in a loop. As this version doesn't need the GPUs, the maximum amount of threads is not determined by the number of cards but by the number of available processors. This is the kind of approach that is normally used in a standard cluster.

We have tested this code in a machine with only shared memory communication. Results for a 210x210x210 grid and the FitzHugh-Nagumo model are presented in figure 2.28. For the maximum amount of processors in the machine, 10, the execution time is longer than 4 seconds when a much smaller grid is used, see the right hand side of figure 2.28. If we extend the behavior presented in the plot an extremely large number of processors would be required to achieve the same speed up as in the GPU case. Probably, a machine with such a number of processors and only shared memory is impossible to find.

To extend the comparison, the execution time as a function of the number of cards is also presented in figure 2.28. These results are shown for 2 different configurations, the first is a 210x210x210 grid, which can be compared to the experiment with only CPUs, and the second is for a 308x308x308 grid as the one used for the results in the previous section. Both experiments were done in just one machine of the cluster (only shared memory communication) and only the amounts of cards for which the domain can be equally divided were used. Already with 5 cards the difference in execution time for the small grid with and without GPUs is huge. In both cases an increase in the number of GPUs provides a faster solution. The shape of both curves is exponential and a limit to the possible speed up is already seen in the 210x210x210 case. This is similar to the effect described in section 2.1.4. For the bigger grid the limit is still not reached with just one computer.

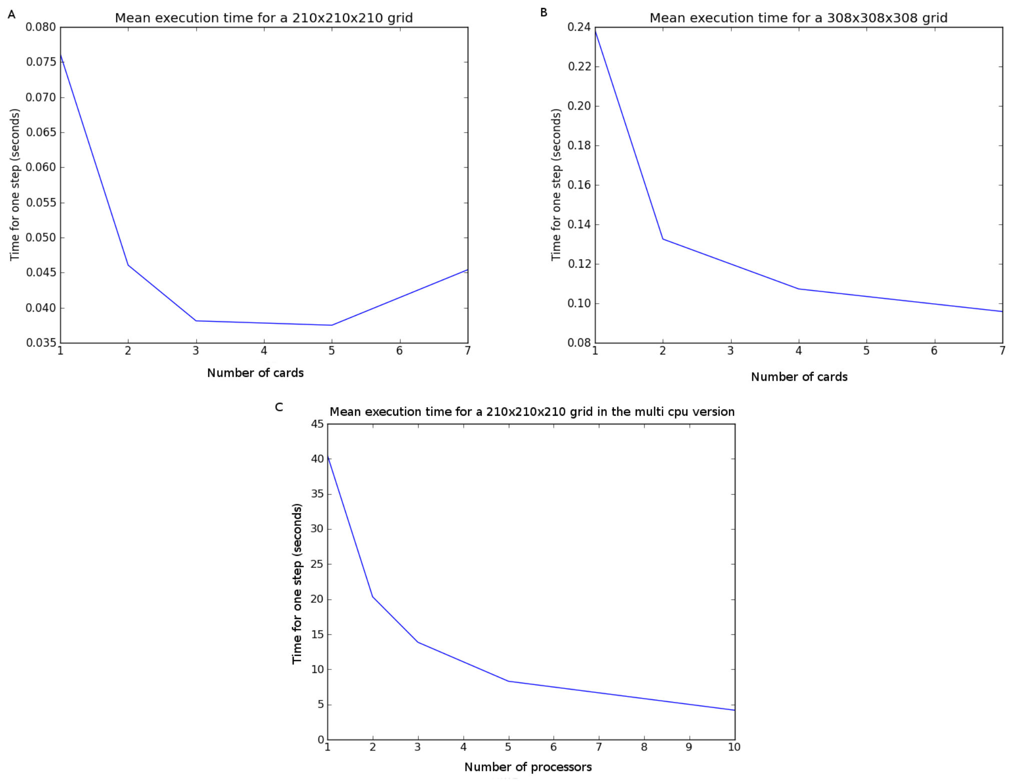


Figure 2.28: A: mean execution time for one time step for a 210x210x210 grid for different number of GPUs. B: The same as A but for a 308x308x308 grid. C: Mean execution time for a 210x210x210 grid as a function of the number of processors when no GPU is used

This difference in speed was as expected, mainly because the computational power of the GPUs is much bigger than the one of the 10 CPUs. Just one card has a theoretical maximum of 1.03 Tflops while the 10 cores have only 0.1064 Tflops. One GPU has much more computational elements than the 10 cores, in fact, the cards are almost 10 times faster. This difference is only with one card and not with the 14 available. Clearly, comparing the 2 hardware is very unfair, but we would need around 100 cores to reach the same theoretical peak of just one card. We don't have access to a machine with enough cores to reach the same theoretical flops of all the card working together, so a fair comparisson is impossible.

Although we are comparing machines with different hardware, the financial aspect should also be considered. Buying a machine with the same computational power as our GPU cluster would be very expensive. The two different types of hardware we are comparing are in a low price range for standard supercomputers. Clearly, for our problem, using a GPU cluster is faster than using a multi-processor machine of a similar price.

In a second experiment our solution was compared to an implementation using the PETSc library [Balay 2012b, Balay 2012a, Balay 1997]. This library contains methods for solving partial differential equations using parallel machines. The current version even provides some support for GPUs. We have used the explicit Runge-Kutta method provided in the library and 98 processors, evenly split between 2 machines. The mean execution time for this version with a 308x308x308 configuration is 3 seconds, which is around 60 times slower than our GPU implementation with the 14 cards (0.06 seconds).

The PETSc implementation first creates a distributed array object to store the results. This is a data structure provided by the library that distributes the data equally between processes. The communication of the boundaries between processes doesn't need to be written by the programmer, it is managed automatically by the library. Each time the right hand side of the equations is computed the values of this array are updated by PETSc. In fact, the only things that the time stepper methods implemented in the library requires are a distributed array and a pointer to a function that computes the right hand side.

Our implementation provides to the library a function that computes the right hand side of the equations. This function is then called by the chosen numerical method for time integration. The function first, reads the distributed array and obtains the boundaries. This is done with one call to a PETSc function. Then, it computes the mean value of y ($\int yP(V, w, y)dVdwdy$) of the points assigned to the process that made the call. This value is then shared with the others via the MPI broadcast function. Normally, the right hand side functions in PETSc implementations do not use directly MPI calls because all the communication is provided by the library. In our case avoiding explicit MPI calls was impossible due to the integral in the equations. Once the final value of the integral is known, a for loop computes the right hand side of each point assigned to the process.

PETSc is designed to work with a set of MPI processes. For this reason, all the communication of this implementation is done via message passing. In the previous implementation (with GPUs) only 2 MPI processes were created, one per machine, and the local communication is performed through shared memory. This avoids the packaging procedure require by MPI and PETSC.

The test was done using 2 nodes of a Dell R815 cluster. Each node has one Intel quad Opteron processor processors with 48 cores running at 2.2GHz. Each machine has 256 Gb of RAM memory. They are connected via infiniband technology

A preliminary version of these results was published in [Baladron 2012a]. There the speed up for an older version of the code is reported. The main difference between the version in the paper and the one on this thesis is how the Runge-Kutta 4 method is implemented. In the previous version the GPU only computed the right hand side of the equations, but not the input for the next step of the method. After each computation in the GPU, the complete result was sent to the CPU which prepared the next call by using the Runge-Kutta formulae and then sent the data back to the GPU. In the current version both the input for the next step and the values of the right hand side are computed on the same kernel, in the GPU. For this reason only the values at the boundaries need to be sent to the CPU instead of all the results. The final amount of memory copied between CPU and GPU is much smaller in the version reported in this thesis. Also, in the previous version the final values of a time step were computed in the CPU, as all the values were already there after previous exchanges of data. On the new version this is done in the GPU as the data is only maintained there and not in the CPU.

2.2.5 Discussion

We have shown two different options for the simulation of the kind of network described in section 2.1.1: Monte Carlo simulations and solving the Focker-Planck equation. If the network size is small enough, doing Monte Carlo simulation would be faster than solving the partial differential equation. This is because the grid size required for solving the PDE is large (300x300x300), so it still requires a lot of instructions. Also, the solution of the Focker-Planck equation is a correct aproximation of the probability density when the number of neurons tends to infinity, so for a small network the behavior of the solution of the PDE may be different from the real dynamics of the network. Instead, for big networks the Focker-Planck equation will converge to the correct probability density, independent of the number of neurons. For this reason, it is always possible to find a number of neurons for which the Focker-Planck equation will be faster than Monte-Carlo simulation. As our objective was to look at the behavior of large network, we have chosen to solve the PDE instead of the Stochastic Differential Equations.

The previous results show the existence of a stationary probability density for a network of noisy FitzHugh-Nagumo (see figure 2.10 and 2.11) or Morris-Lecar neurons (see figure 2.22), a fact which might have several implications for the way the brain may encode information. In a network with the propagation of chaos effect each neuron is an independent unit which after convergence follows the same stationary law. As the state of each cell is a sample of the process, a neuron in a different population may see as many samples as its number of synapses. If the number of connections of this postsynaptic cell is big enough it may produce a sample based representation of the probability density containing all the information from the previous population.

The time needed for gathering enough samples to produce a good approximation depends on the number of synapses. As soon as the convergence is achieved each neuron represents a sample, so if for example a neuron has 10,000 synapses, after the stationary solution have been reached it will immediately have 10,000 samples. This is why in a dense network the time needed for conveying information between one area and another may be limited by the time necessary for convergence.

The results shown in this section indicate that this convergence speed may be tuned by 2 factors: external noise (see figure 2.13) and input (see figure 2.18). When both values are high the convergence is fast and when they are low it is slow. Neurons have to deal with high levels of noise in the brain ([Knoblauch 2005, Faisal 2008]) so the only options they have are to establish mechanisms to reduce the disturbance it produces so that they can extract useful information from incoming signals or transform this inherent characteristic of the system into a tool to improve its efficacy. The existence of the propagation of chaos effect and the possibility to increase the speed of the system by changing the parameters is an argument in favor of the second option. In fact, the hypothesis that neurons are grouped in populations as a way to face the abundant noise is not new (see for example [Shadlen 1998, Hoch 2003]). Our approach is a new point of view on how this can be done using the propagation of chaos effect.

The different noise sources are able to modify the membrane potential of a neuron in different ways. They may even be able to make the cell generate an action potential which would not have been produced if noise was not present. High levels of noise would produce a large amount of undesired spikes. This means that if we assume that the brain works in the presence of large amounts of noise, neurons will always be generating spikes. As the deterministic input value in the model represents the sum of all the activity from other brain areas, this value will include all these undesired spikes. Due to noise, neurons will hardly be quiet making this quantity larger than 0 and probably large enough to fulfill the requirements for fast convergence.

Another point of view for the encoding of information in neurons is that everything is contained in the firing rate. One cell that represents a certain property will spike more if its preferred input is present in the environment. The problem with this approach is that each postsynaptic cell will need to compute a temporal average in order to read the message from the presynaptic neurons, requiring time to do so. This may be too slow considering that there exist several processing steps before we can react to a change in the environment, especially if reaction time to images in humans have been estimated to be around 400 ms ([Thorpe 1996]). For more details on the advantages or disadvantages of firing rate codes see [Gerstner 2002].

In the approach proposed here neurons also have to wait a minimum amount of time for a computation to be performed as the probabilities need to converge. The main advantage of this new proposal is that this time can be tuned, as we mentioned before. In the case of the firing rate there is no parameter that can be tuned to reduce the waiting time.

If the stationary solution was not used by the brain to encode information its appearance would indicate the end of the useful time for a neuron. In this case all the information would have been encoded in the structure of the changes of the probability density, which no longer occur as soon as convergence is reached. The simulations shown here indicate that this convergence is fast if the parameters are well set, so under these circumstances the brain would need to be extremely fast not to lose information.

This hypothesis is supported by the final experiment where the input value was changed (see figure 2.20). In this simulation the system shows the ability to go from one stationary solution to another even faster than before. This shows how a system that uses a sample based representation may work, once a stationary distribution is achieved this information is automatically passed on to the next population and new computations may begin.

The experiments also show that large modifications in the synaptic noise do not considerably modify the solution (see figure 2.19). This is another interesting effect of the network as it is able to deal with significant disturbances at the connection level. The shape of the probability density changes in these cases but doesn't affect the marginal probability density over the variables V and w, giving the same firing rate and mean voltage variable.

2.3 Multi population Fokker-Planck equation

In all of the previous experiments we had only one population of neurons. In these models the weights depend only on the pre and post synaptic population; all of the synapses were the same. This means that no complex connectivity patterns occur in the simulated network. In this section we introduce more populations and hence more complex networks.

In the simulations of this chapter each population is described by a different, but coupled, Fokker-Planck equation. The computational complexity is increased from the previous simulations as the total number of points to be updated is now multiplied by the number of populations. Clearly, the amount of necessary operations and memory can increase for a large number of populations, enough to make simulation unfeasible.

We show in this chapter simulations for two different multi-population networks. Experiments with a simple two population model of a rat barrel cortex are presented first and then a model similar to the ring model (see Chapter 1) of a V1 hypercolumn. Although in the second case we already had hardware limitations we show how the GPU cluster may provide us with results that require a huge computational power only comparable to that of a standard cluster with hundreds of Intel like processors.

2.3.1 Implementation issues

We have extended the code developed for the previous experiments to be able to manage several populations. The first step in this new version is to distribute the populations among the cards. This is done in such a way that all the resources for the same group of neurons are located in the same machine. Then the points in the grid for each population are divided equally between the GPUs assigned to it (amount smaller than the total number of cards).

This scheme allows a fast communication because the coupling between populations is only in the integral in equation (2.11) ($\bar{y}$). Each population doesn't need to know the values at all the points of each of the other groups to which it is connected but only requires the mean value of y for each of them. This quantity is computed locally by the resources assigned to a population and then the result is broadcasted to the rest of the processor-card pairs. The amount of information that needs to be sent through the network (the bottleneck in this kind of application) is equal to one floating point number times the number of populations assigned to each computer. This is an extremely low number compared to the previous implementation where all the boundaries need to be transferred.

The new communication scheme reduces the increase in computational time required by the bigger amount of points. This is specially noticeable when a small number of populations is used (as in the barrel cortex model to be described next) as the difference in computational time is not extremely big when a new population is added and numerical experiments are still feasible. A special case occurs when each population is assigned to just one GPU (number of population = number of cards). In this situation, there is no need to share boundaries between processors, requiring no memory copy between GPU and host. As mentioned before only the integral value is needed. This reduces computational time, but as this occurs only for a large enough number of populations (as in the hypercolumn model that is presented later) the total amount of points to be updated is still very high, making the simulations still very long. Figure 2.29 shows a diagram of the distribution of points between the 2 computers and the messages they need to send each time the right hand side is computed. Also, figure 2.30 shows a flow diagram of the process that can be compared with the one in figure 2.9. On the new diagram the number of lines that cross between the GPU and CPU areas is smaller and also the amount of data sent on each crossing is reduced.

2.3.2 Two population network: a barrel cortex model

The main tool that rats and mice use for building an internal map of their environment is not the visual system as in humans. These animals mainly use their whiskers which are highly sensitive sensors that provide them with information about the objects and textures around them. The area of the cortex in charge of integrating the information from the sensors is called the primary somatosensory cortex.

The somatosensory cortex is also called the barrel cortex due to the existence

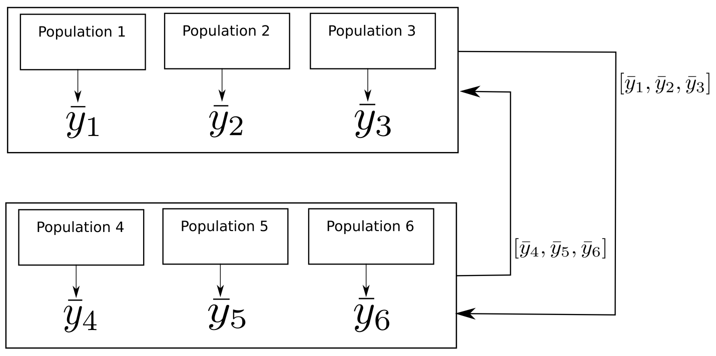


Figure 2.29: Diagram showing the distribution of populations in a multi-population experiment

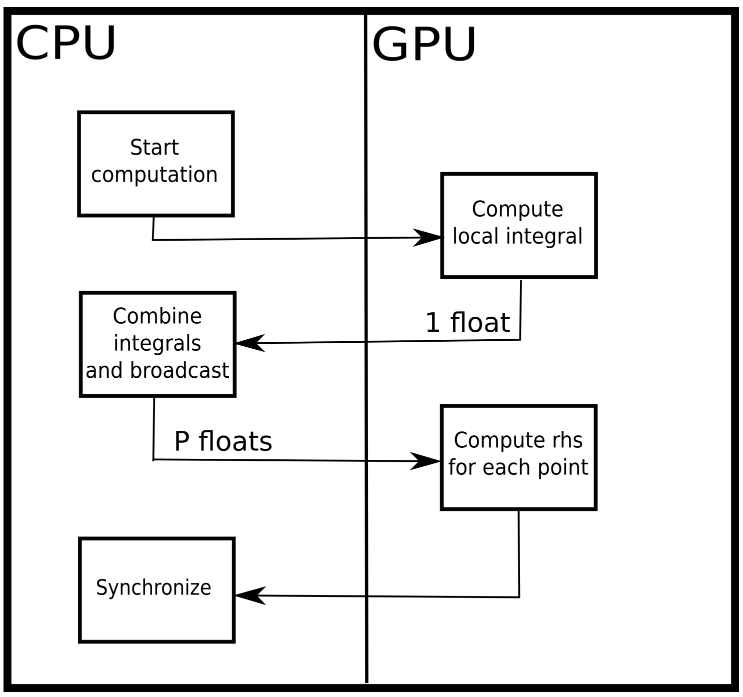


Figure 2.30: Flow diagram of the procedure for the computation of the right hand side for the multi-population experiments

of discrete structures of neurons, or barrels, that represent the different whiskers. These groups are organized in a similar way as the whiskers are in the snout. The deflection of each whisker will activate its corresponding barrel. A diagram of this structure is presented in figure 2.31. For more information about the barrel cortex structure and connectivity to other areas of the brain see [Petersen 2007].

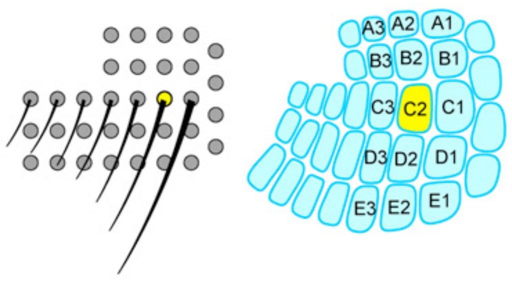


Figure 2.31: Diagram representing the distribution of barrels in the somatosensory cortex. Each blue element correspond to a barrel which respond to the whisker at the same position. The standard nomenclature for some of them is also shown. Adapted from [Petersen 2007]

A computational model of a barrel consistent with biological data is proposed in [Kyriazi 1993]. The authors use 100 neurons, where 70 are excitatory and 30 are inhibitory. The model is based on the following four organizational principles: inhibitory and excitatory neurons present different nonlinearities, the input coming from the Thalamus is the same for both types, the connections are among and between cells of both types and inhibitory neurons are more responsive to input. The response of neurons in the simulations were similar to the ones found in real barrel neurons.

A reduction of this model was proposed in [Pinto 1996] where the full system is described by just 2 equations, one representing the average activity of the excitatory population and another for the average activity of the inhibitory population. All the parameters for this simplified version can be obtained from the original model.

The reduced model was then used in [Pinto 2000, Pinto 2003] to study the response of the system to input with different velocities and amplitudes. One of the main predictions of their simulation is that the system is a temporal contrast detector, i.e. it responds selectively to rapid changes in the input. The network is more sensitive to the speed in the change of the input function than to its magnitude.

The authors propose a simple numerical experiment to prove their hypothesis. First, two input functions with a triangular shape are created. Both of them reach the same peak value and afterwards decrease to 0. The main difference between them is the slope of the lines as one of them reaches its maximum value before. An example is shown in figure 2.33A. These two input functions, are used in different simulations. If the system is sensitive to the magnitude, the output in both cases should be the same but slightly shifted. If the hypothesis of the authors is right then the output should be higher for the faster triangle. In fact, this is what happens as the activity is higher when the peak is reached faster.

When a fast input is presented to the system, excitatory cells emit at least one spike at short latency. This increases the activity in the population which is raised even more by the recurrent connections. After a few milliseconds this activity reaches the inhibitory population through the synapses. Inhibitory cells start to spike and their activity overwhelms the excitatory response reducing the response of the network. If the input is slower, inhibitory neurons will have more activity in comparisson to the other population at the beginning of the process (due to a smaller amount of spikes coming from the other population and their faster response to the input) inhibiting the possible explosion of activity on the excitatory cells

We have recreated the model but using our mean field approach. Each of the two populations in the original network is represented by a different Fokker-Planck equation. The synaptic weights between the population were taken from [Pinto 2000]. Also, to follow the original model and fulfill biological constraints, different time constants were introduced for the two populations to assure that the inhibitory cells respond faster to the input. The input received by the two populations is also different, as it is multiplied by different constants. The inhibitory neurons receive larger input values to enhance the fact that they should respond faster. A diagram with the final structure of the network and the weight values is presented in figure 2.32. This system of partial differential equations was solved using the strategy already described. Figure 2.33 shows the two different inputs used by the system and the mean voltage in each of the two cases. Only the mean was measured as it is the value computed in the original analysis made with the reduced model. The output for the faster input reaches a higher peak supporting the temporal constrast detector hypothesis. The difference between the two outputs may be enhanced if the difference in slope of the inputs is increased.

The only difference between our version of the model and the work by Pinto is the different type of neuron model. The results show that the same effect he has found for integrate and fire type of neurons can also exist with conductance-based models. The main objective of the experiments described before was to test our new mean field reduction with a small multi-population model with a known behavior. Experiments were succesful as we could reproduce the dynamics, but using a different set of equations. This allows us to extend our approach to larger network and produce new models as the one which is described in the next section.

2.3.3 An orientation selectivity model

One approach for modeling the selection of edge orientations in the primary visual cortex is to consider one hypercolumn as a collection of connected elements representing the different possible angles in one receptive field. The connections depend on the difference between the preferred orientations. The Ring Model, described in section 1.2.4.2, is an example of this approach. In this case the spatial variable of the neural field equations represent a continous set of populations with different

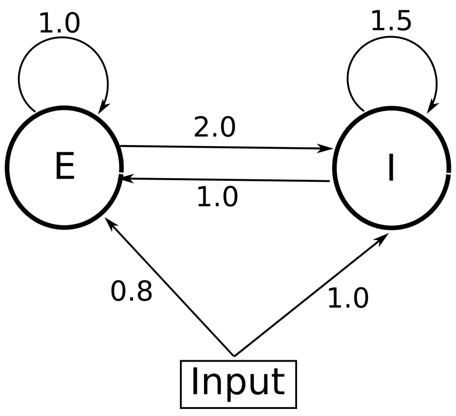


Figure 2.32: Structure of the two population network representing one barrel from the somatosensory cortex. The numbers represent the weights between the elements.

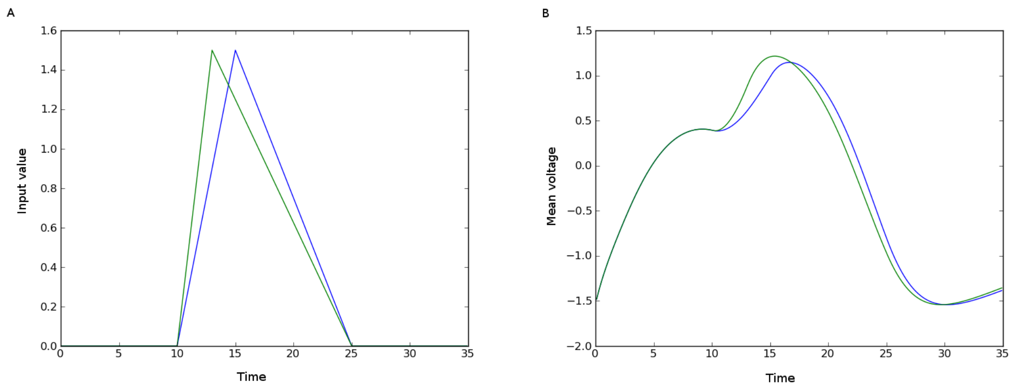


Figure 2.33: A: The 2 different input used for the barrel model two population network. The two reach the same peak but with different speeds. B: The mean voltage when the two inputs are applied. The colors correspond between the two plots.

prefered orientations.

A different approach is taken in [Battaglia 2011] where each orientation preference is represented by one, discrete, neuron instead of the population measures of the Ring Model. The author propose to describe one hypercolumn with two interconnnected sub networks, one representing layers I to IV of V1 and the other layers V and VI. Each of these groups is composed of excitatory and inhibitory neurons diving each of the two networks into two populations. One angle, between -90 and 90, is assigned to each cell. The connections are random with a probability that depends on the orientation preferences of the 2 cells and on their population (excitatory or inhibitory).

We propose a different model that combines these two methods and uses the mean field approach described in section 2.1. As in the model by Battaglia each orientation preference is represented by one excitatory and one inhibitory element. The main difference is that both of them are populations of neurons (as proposed in the Ring Model) instead of simple cells. Each population is approximated by its mean field limit corresponding to the Fokker-Planck Equation.

The connectivity of our new approach also depends on the orientation difference and on the type of population. Excitatory populations connect strongly to similar elements while inhibitory populations have strong synapses with disimilar cells. We use the same synapse parameters and reverse potentials as in [Battaglia 2011], given in their detailed neuron model.

The input depends on the preferred orientation and represents the initial edge detection done by the first areas of the visual system. The system is expected to improve the initial selectivity by providing an output with a sharper shape than the input coming from the retina and the LGN. This improved tuning is due to the effect of lateral connectivity.

A similar network structure has been used by Rolls and Deco [Rolls 2002, Rolls 2010] for studying attention, they call it a competitive network. In their case each excitatory population represent one possible decission. They are also connected to inhibitory pools wich creates a competition by reducing the activity of possible contradictory decisions. They show that the system is able to maintain a certain level of activity in just one population, indicating the final decision made. Also they represent attention as a second input to each population that biases the final decision.

Figure 2.34 shows a diagram of the structure of a network with six populations. Each colum of the figure is composed of an excitatory and an inhibitory population corresponding to the same preferred orientation. The connectivity structure shows that we have removed the connections between inhibitory populations that do exist in the model of [Battaglia 2011]. The labels of each arrow are the different weights used in the simulations that will be described next. The output of the system is represented by the activity of the different excitatory populations.

Simulating a network like the one in figure 2.34 requires solving a system of Fokker-Planck equations. This is much more complex than solving a single equation (see section 2.2.1). As described in section 2.3.1 the populations must be distributed

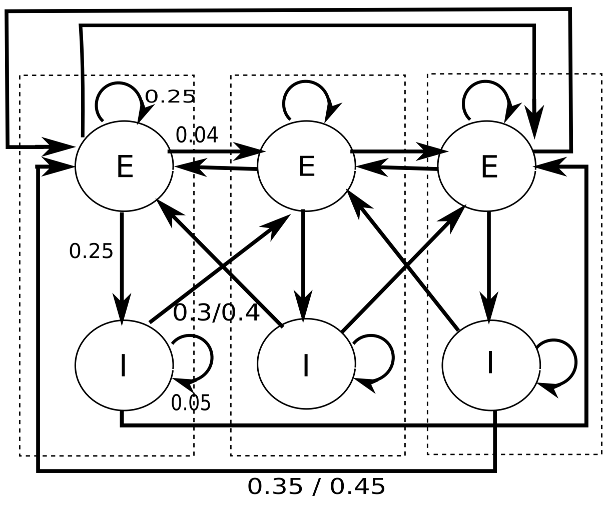


Figure 2.34: Diagram representing the structure of our model for 6 populations (3 orientations). Each circle represents one population in the mean field limit. The label next to the arrow show the weight between populations. For the inhibitory to excitatory connections the two numbers correspond to weights used in different experiments.

among the possible cards in such a way to minimize the amount of data transfered. The hardware in which we have performed the simulations is fixed, so, when we include more populations each pair GPU-processor will get assigned a larger amount of points. For example, if we want to solve just one Fokker-Planck equation in a grid of $308 \times 308 \times 308$ in a machine with 7 cards, each GPU will need to compute the right hand side of $(308 \times 308 \times 308)/7 = 4,174,016$ ordinary differential equations. If, one more population with the same grid size is added, each GPU will be assigned with $2 \times (308 \times 308 \times 308)/7 = 8,348,032$ points.

Our implementation creates one thread per point assigned to the GPU, so, when more populations are considered a larger amount of threads are required. The total number of threads depends not only on the amount of populations but also on the size of the discretization. The grid structure we have used in the one population experiments is already very big and the total number of points used in those simulations is larger than the maximum number of threads than can be created in one GPU. This maximum is given by hardware restrictions and depends on the amount of registers required by the kernel that is going to be executed. For this reason, the ordinary differential equations coming from the discretization of each population needs to be solved by at least two cards.

As two cards are required for each Fokker-Planck equation, the maximum number of populations we can simulate with our hardware is six. In this case, three populations will be assigned to each machine, using only six of the seven available GPUs. The extra GPU in each machine will not be used because it can't solve by itself a complete population. Dividing one population between two cards on different machines will require more data to be transferred through the network, making the simulation much slower. If this is the case, the implementation could not benefit from the fact that the equations only depend on the mean of the y variable to reduce the communication (see 2.3.1).

We have performed several numerical experiments with this network. The reverse potentials were chosen by normalizing the values proposed in [Battaglia 2011] to the spike size of the FitzHugh-Nagumo model. This is necessary because the original values were proposed for voltages between -60mV and -60mV while the FitzHugh-Nagumo only varies between -4 and 4. The reverse potential for excitatory synapses was set to 5.4 and for inhibitory to -3.1. The value of the parameters a_r and a_d of the synapses were also taken from [Battaglia 2011] and are 1.0 and 0.33. For the first group of simulations the input is 0.8 for the first 2 populations, 0.2 for the second pair and 0 for the final group. This means that the edge presented to the eye has an angle close to the orientation preference of the first group of populations, but not very far away from the preference of the second. The system is expected to integrate the information from all the populations and select just the one that is closer to the input angle.

The first experiment was done with an external noise of 0.45 and shows how the output of the system is a very good representation of the input function. This can be noticed in the firing rate of each of the excitatory populations as it is shown in figure 2.35. The rate for the one with larger input is higher than all the other ones. The final value of the highest rate is twice as big as the second, so no real sharpening is occurring. If the system was behaving as expected the ratio between the two outputs should be bigger than between the two inputs. In a perfect scenario the activity of the second group of populations should be very close to 0, thereby selecting the first orientation.

Although the system was not sharpening the initial angle selection, it was creating a much better representation of the input than what a group of isolated population could do. This effect can be noticed by comparing the rates shown in figure 2.35 with the ones in figure 2.36. This last figure shows the results of the same experiment but removing all the connection between different populations (leaving only connections where the presynaptic and postsynaptic neuron belongs to the same population). This last plot shows how without lateral connectivity the rates are very close to 0 and the activity of the three excitatory populations converge to the same value. The effect of lateral interaction seems to be critical in a sensory system working in the mean field limit.

A raster plot showing the activity of 1,000 neurons of each excitatory population is also shown in Figure 2.35. The amount of spikes is bigger for the populations with higher input. The population with 0 input still spikes due to the amount of noise, although it has less activity than the others. The initial high amount of spikes in all the cases is due to the initial conditions which are described by a Gaussian probability density with a mean outside the limit cycle of the isolated neuron. Once the simulation starts the mass tends to move towards the limit cycle of the isolated

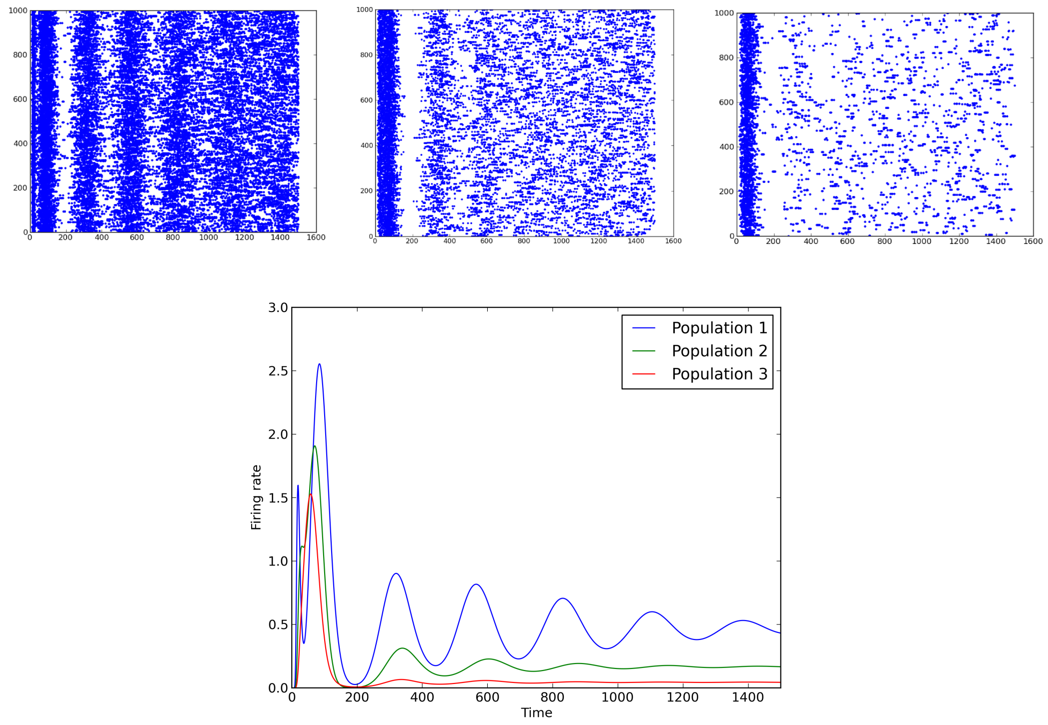


Figure 2.35: Results of the first six population experiment. Each raster plot in the figure corresponds to one of the three excitatory populations of the first experiments. These results are with noise level 0.451 and low level of inhibiton (0.3). The value of the input is decresing from left to right. Below the raster plots is the firing rate for each case. Movies for this simulation are available in the web site

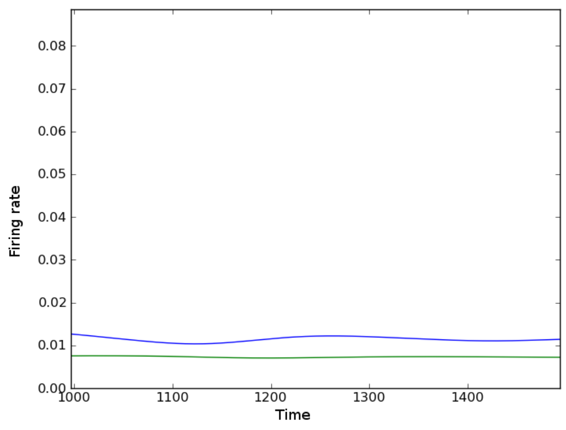


Figure 2.36: Each of the lines of the plot correspond to a simulation of a one population network with the same parameters as each of the excitatory populations on the ring model experiments. The main difference is that the lateral connections were removed. The blue line corresponds to the population with the highest input and the green to the other two populations, which at the period of time shown in the plot had the same firing rate

neuron, crossing the threshold during its trajectory. A movie with the evolution of the probability density for each population can be found in the web page.

The raster plots of figure 2.35 were generated using the solution obtained from the Fokker-Planck equations. The different probability densities were used to compute samples of the mean field process by solving the stochastic differential equation ((2.10)) using the Euler-Maruyama method. Each of these samples is considered as one possible trace for a neuron in the network. This is fast to compute as it is equivalent to solving the equation for just one neuron. After this, to each voltage trace, the same threshold as before was applied, if this value was passed from below a spike was generated.

In a second experiment the noise level was reduced. When the same simulation is run with noise level 0.27, in each population there is oscillating activity as can be seen in figure 2.37. The rates shown in the plots seem to be periodic with intervals where the value is very close to 0. This shows that most of the neurons are spiking very close to each other, switching between periods of high activity and periods of low activity. The raster plots, also shown in the same figure, confirm this, as on each one there are vertical layers of spikes, specially for the first population. Movies for this simulation can also be found in the web page.

As in the previous experiments the rate for the population with the highest input is much bigger than the others. The second population still has activity, also periodic, with peaks at similar time instants as the first group of neurons. This shows that the output is still a better representation of the input than in the case of isolated populations.

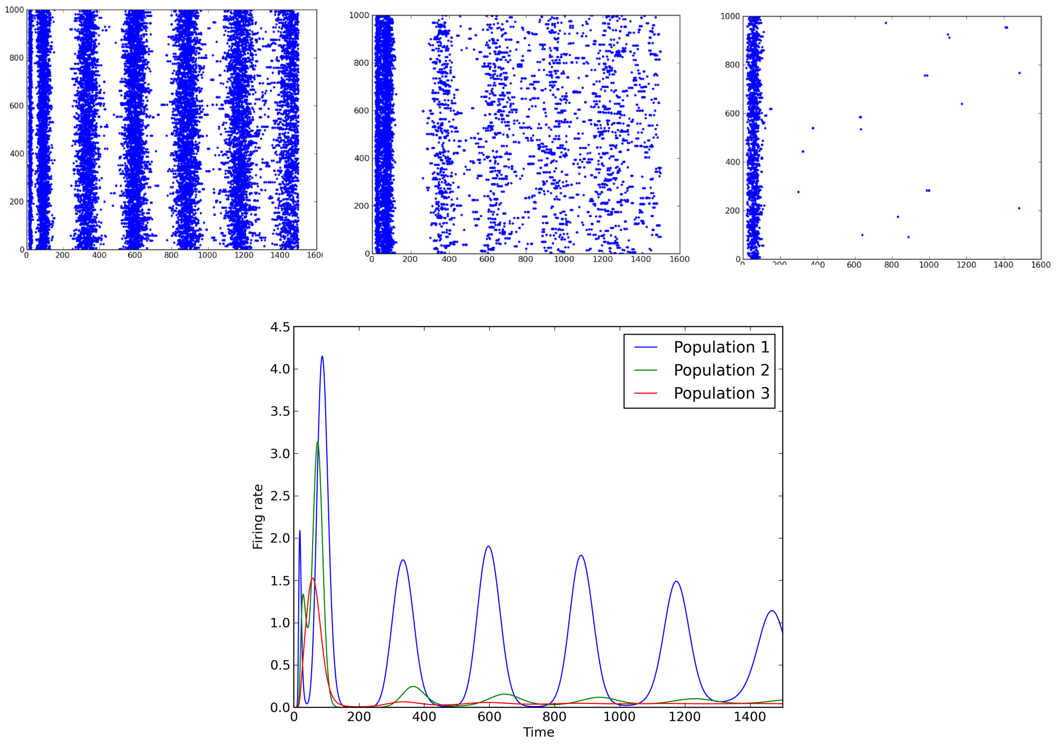


Figure 2.37: Results of the second six population experiment. Same as in figure 2.35 but with a smaller level of noise (0.27). Movies for this simulation are available in the web site.

In order to reduce the activity of the second population, more inhibition is needed as was shown by a third multi population experiment. In this case, the inhibition was increased, leaving all of the other parameters equal (noise 0.45). All of the rates, as shown in figure 2.38 were reduced, even the one for the population with highest input. The main difference is that now the activity of the first group of neurons is much higher than the rest. The rate at the final time step is 8 times higher, showing a sharper output, fulfilling the expected behavior of the model.

It is easier to notice the difference by looking at the raster plots shown in figure 2.38. Here the amount of spikes on population 2 (after the activity due to the initial conditions) is even lower that the one of the population with no input in the previous experiments. It is also very similar the output for input 0.2 and input 0. By looking at the amount of spikes it is possible to notice how the first angle was clearly selected.

When this level of inhibition was maintained and the noise level reduced to 0.27, the oscillatory activity appeared as in the second multi-population experiment. Figure 2.39 shows the results in a similar way as for the previous cases. The rate for the population with bigger input has oscillations while the two others are reduced to a value close to 0. This is also a signal of sharpening but only if the activity is read at the peak of each period.

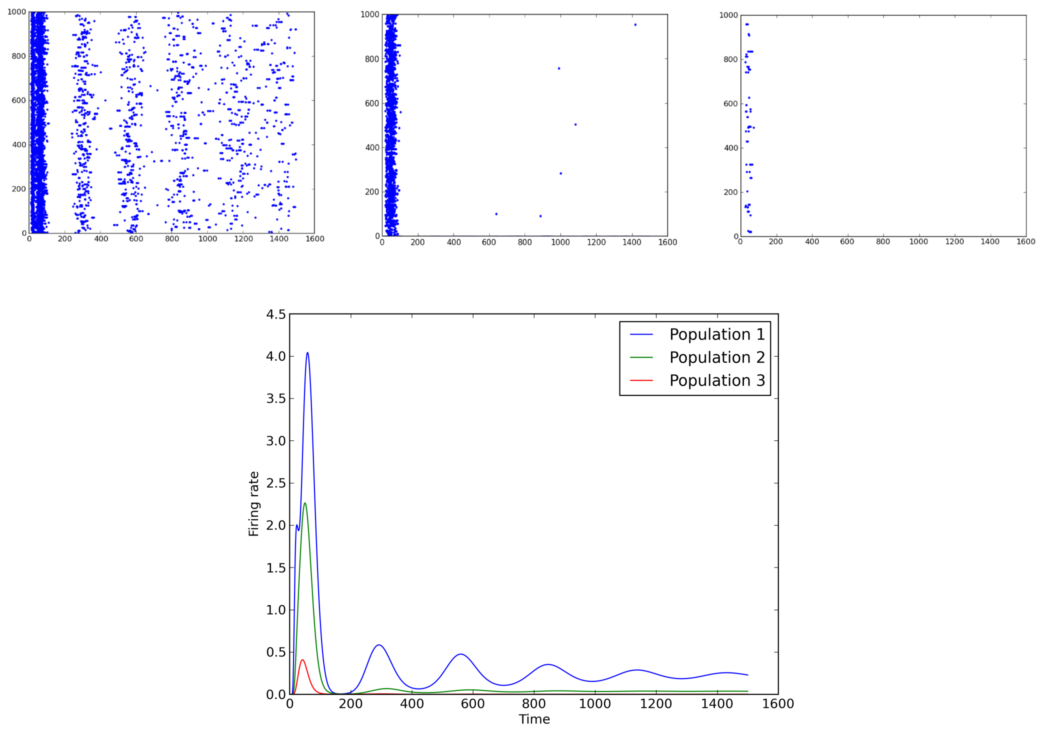


Figure 2.38: Results of the six population experiment with higher inhibition (0.4) and noise level 0.45. Movies for this simulation are available in the web site.

In order to check the sharpening capabilities of the network (with the second

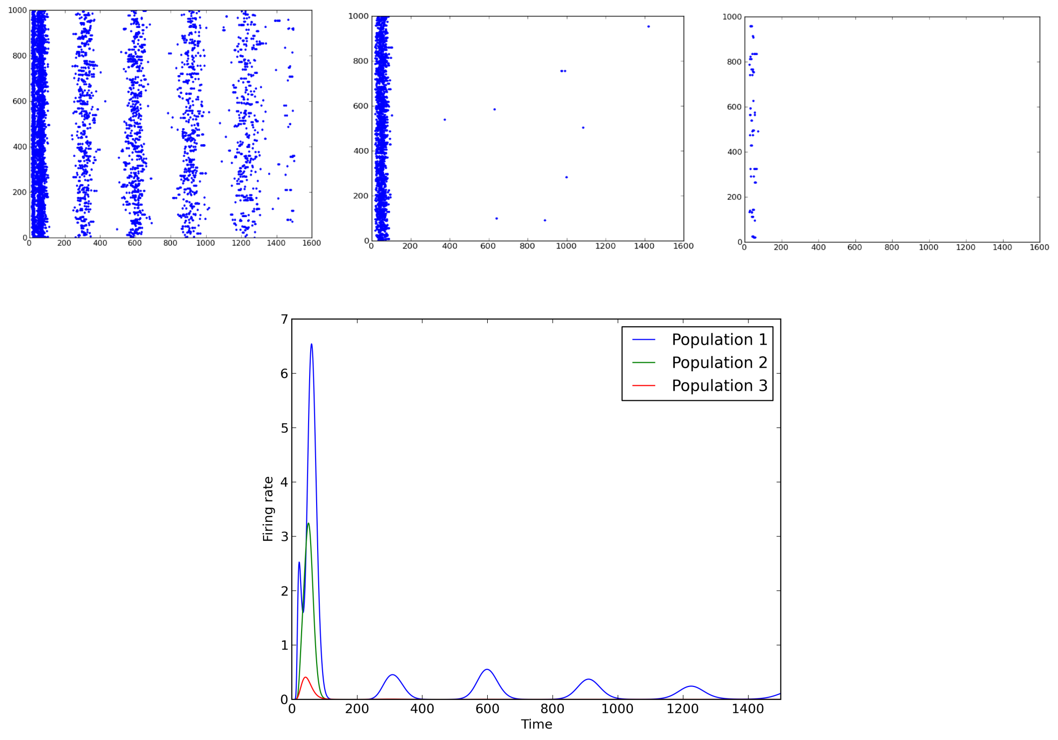


Figure 2.39: Results of the six population experiment with higher inhibition (0.4) and noise level 0.27. Movies for this simulation are available in the web site.

set of weights with enhanced inhibition) a second group of numerical experiments were performed. For these simulations the input function was changed, now, for the first and the third pair of populations this value was set to 0.1 while for the second pair it was changed to 1.0. These quantities represent an initial selection equal to the orientation preference of the second population. The network should enhance this initial approximation by reducing the activity of the first and third group while increasing the second.

The results of these experiments are presented in figure 2.40. The two plots on the figure show the firing rate when the noise level is 0.45 and 0.27 respectively, the same two values used previously. On the two cases only the population with the highest input is spiking and it clearly represent the selected angle. As the system has destroyed the activity on the other two populations it is really enhancing the initial selection. When the external noise level is changed, the same effect than before can be observed. For high levels of noise the firing rate seems to be converging to a stationary value while for low levels there are oscillations.

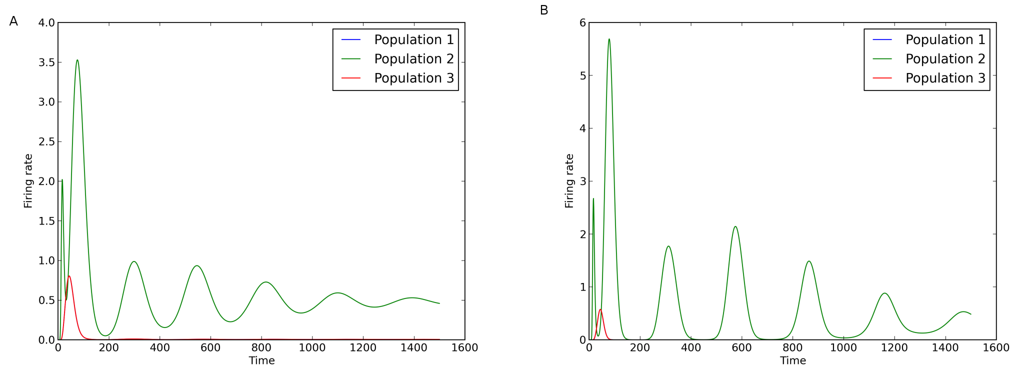


Figure 2.40: Results of the second group of six population experiment with a different input function. A high input value is presented to the two second populations and a smaller one to the others. A: experiment with noise level 0.45. B: experiment with noise level 0.27

The numerical experiments with the first input function were repeated with the Morris-Lecar model. This was done for the same reason as for the one-population simulations. The connection structure was kept as in figure 2.34 but the values were adjusted and reduced. This reduction is neccesary due to the difference in the height of the spikes in the two models. The connectivity values are shown in table 2.5.

The results for this experiments are shown in figure 2.41. There the V-w marginals of the final stationary solution of each of the three excitatory population are shown. For populations 2 and 3, the two with the lower input values, all the mass of the probability density is distributed around a peak on a negative voltage. This means that all the neurons keep a voltage value close to a negative potential, the resting state. For the first excitatory population the probability mass is distributed along the limit cycle, indicating that some neurons do emit action

	0	1	2	3	4	5
0 (E)	0.08	0.0	0.01	0.12	0.0	0.14
1 (I)	0.08	0.015	0.0	0.0	0.0	0.0
2 (E)	0.012	0.12	0.08	0.0	0.012	0.12
3 (I)	0.0	0.0	0.08	0.015	0.0	0.0
4 (E)	0.0	0.14	0.012	0.12	0.08	0.0
5 (I)	0.0	0.0	0.0	0.0	0.08	0.015

Table 2.5: Table with the weights for the multi population experiment with the Morris-Lecar model. Each row shows the connections arriving at each of the populations. The letters next to the population number indicates if its an excitatory or inhibitory group

potentials. There is one peak in this case, but it is located on a high voltage value, indicating that the majority of neurons have large membrane potentials due to their spiking activities. The system has reduced the activity of populations 2 and 3, leaving only population 1 active population 0. This is a sharpening effect as clearly the preference from the first group of population has been selected.

2.3.4 Discussion

The experiments in this section suggest that mean field limit approximations of large populations of neurons may be useful to describe fundamental mechanism of cortical activity. This is the contrary of the classical grandmother cell hypothesis were just one neuron encodes the information [Gross 2002]. The idea of populations as elemental elements on the brain is not new and an extensive amount of work have been done in understanding how they can encode information (see for example [Pouget 2000, Zemel 1998, Ma 2006]). Normally, populations are consider as encoding not the value of a variable buts its whole probability distribution. Our approach doesn't explain how populations of neurons can encode any given probability function but it can be used to describe the probability distribution of large ensembles of cells. If the populations is encoding a given probability function, the law governing the behavior of the cells must be related to the function being coded. We believe that this relation can be studied and a combination of our approach and population coding theory can be obtained.

The results of the multi-population network provides more information on the effects of noise for this kind of systems. The numerical experiments show that increasing the noise can destroy firing rate oscillations and replace them with the appearance of a stationary firing rate. In both cases the system behaves as expected improving the original angle selection. In the oscillatory regime the peaks in the different population are only slightly shifted, so if the points where the activity reaches 0 are avoided, the difference in the firing rate can be detected. These results reinforce what was found in the previous single population experiments regarding an increase in convergence speed with higher levels of noise.

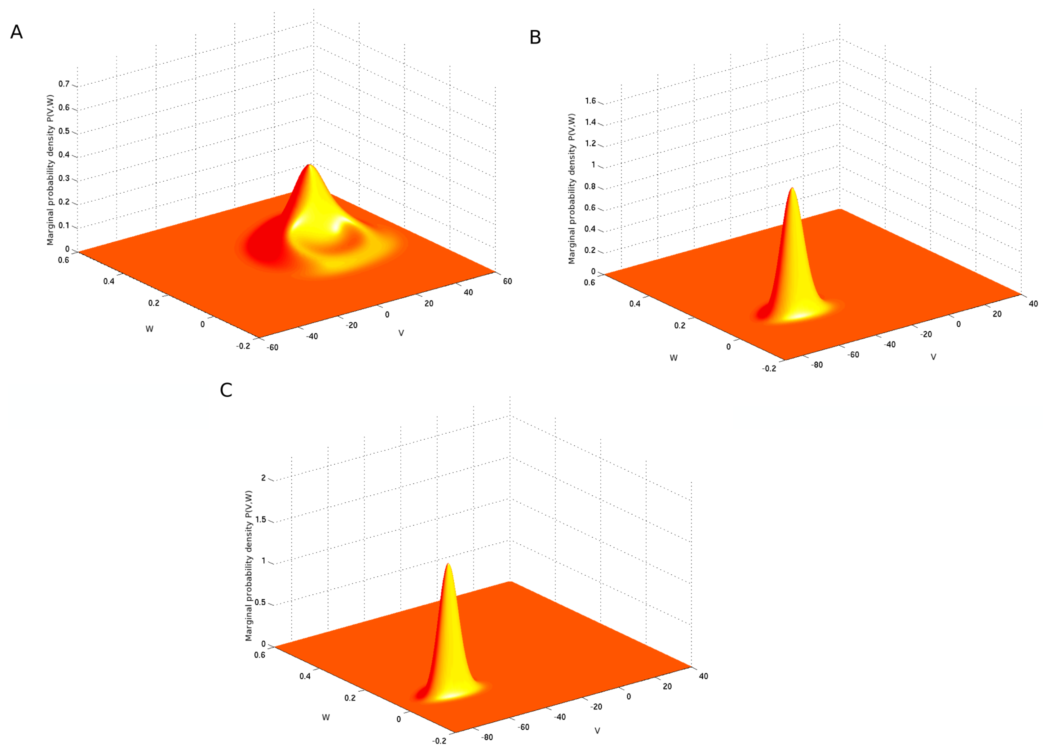


Figure 2.41: Stationary solution for the three excitatory populations of the experiment with the Morris-Lecar model. A: Population 0, with the highest input value. B: Population 2, with a smaller input value. C: Population 3, with input 0. Movies for the three populations are available at the web page.

If the oscillations present in the low noise experiments was used as a tool for conveying information in the brain, a sudden raise in the noise level, due to, for example, a change in the environment, would completely destroy the computing capabilities. As the brain is effective, independent of raises or high noise level we believe that the phenomena of the stationary distribution is a better tool for encoding information and that may be used in some areas of the cortex.

The difference between the rate of the first and third multi-population experiments show that the enhancing capabilities of the system depends on the level of inhibition. This functionality is important because it could allow the brain to choose correctly between a set of possible stimuli. Normally the output of a set of oriented edge detectors (sensors) is too ambiguous to reach a decision, but the sharpening capabilities of a system like the one presented here can improve this situation.

The experiments have shown the crucial effect of inhibition in the enhancing capabilities of the network. Inhibition is the only mechanism the system has for reducing the activity of populations with an orientation different from the input angle. If inhibitory connections are too low too many populations should still be active and the selection will not be clear enough. A correct set of weights may not only reduce the activity of the wrong orientations but also diminish the amount of spikes of the correct population. As have been shown in the experiments, this may still be a correct behavior if the only population that generates spikes is the correct one and the amount generated is large enough to make a clear selection.

The simulations done with the Morris-Lecar model show that the previous results are not strongly depending on the fact that we used the simpler FitzHugh-Nagumo model. They show that a more realistic network, in the case of the multi population experiments, even improves the sharpening capabilities. In the experiments the system was only studied with a high level of noise that produced a stationary distribution as the main objective was to see if with this more realistic model the expected behavior was still achievable.

The approach presented here is more realistic than the one in [Battaglia 2011], on which we based our model. Battaglia uses just one neuron per orientation while we use a complete population. Experiments in the visual cortex have shown that many neurons share similar properties [Mountcastle 1997], as it occurs in our model. Any damage to a cell would change completely the behavior of a network like the one proposed by Battaglia but it will not greatly affect a dense network as the one we use.

Our model includes more detail than the Ring model of orientation (the neural field model which also inspired our approach). By knowing the probability density of the neurons it is possible to generate spike trains for any neuron belonging to the network. This is impossible with the ring model, for which only the mean value of the voltage from each population is known. In a neural field model there is no clear relation between a detailed neuron description and the final equations. In our approach this relation is known and can be exploited to, for example, generate action potentials to be used as input for large scale models of higher cortical areas.

	Explicit	**Implicit**
Δ t size	small	big
System of equations	no	yes
Amount of computations per time step	small	big

Table 2.6: Table showing the difference between explicit and implicit methods for stiff equations.

2.4 A faster but less accurate numerical method

The main difficulty for performing the previous experiments was the stiffness of the equations, which make necessary the use of an extremely small time step (0.001). If this value is increased, numerical errors appear and they get amplified and spread with subsequent time steps. For a long enough simulation, the results diverge and can't be represented in memory by floating point numbers. For this reason, observing the behavior of the network for long periods of time is slow even if the computations for one time step can be done very fast on GPUs. In all the previous simulations we performed 150,000 time steps and going beyond this limit would require either a change in methodology or an even more powerful hardware.

This problem is even worse in the multi-population case, where a smaller amount of computational units are assigned to each population, making the computation of each time step slower. This situation, together with the hardware constraint described in section 2.3.3, limit the maximum amount of possible populations to simulate. One option for avoiding the hardware constraint is to compute each population sequentially. To do this, first the CPU copies the data from one population to the GPU. Then, each card computes the mean y for that population. Once they are finished, the result is copied back to the CPU. The process is repeated for each population. Then a similar loop is executed to compute the right hand side for each point. A flow diagram of the process is shown in figure 2.42. This is a slow process (lots of memory transfers and sequential computations) which, due to the stiffness of the equations, would need to be repeated a large number of times.

There are two different paths to follow when dealing with a stiff equation. The first one is to use a very powerful machine on which computing the right hand side of the equation is so fast that you can reduce enough the size of the time step. This is the approach we were following up to now, and with which we could obtain all the previous results. A second option is to use an implicit numerical method. In this kind of approach at each time step a system of equations (nonlinear in our case) needs to be solved. The unknowns of this system are the values of the function at the next time. This different type of integration scheme can be very efficient in the case of stiff equations (see the book [Hairer 2010] for more information on stiff equations and numerical methods for solving them). Table 2.6 shows a summary of the advantages and disadvantages of each method.

Using an implicit method for the Fokker-Planck equation, with the grid size we

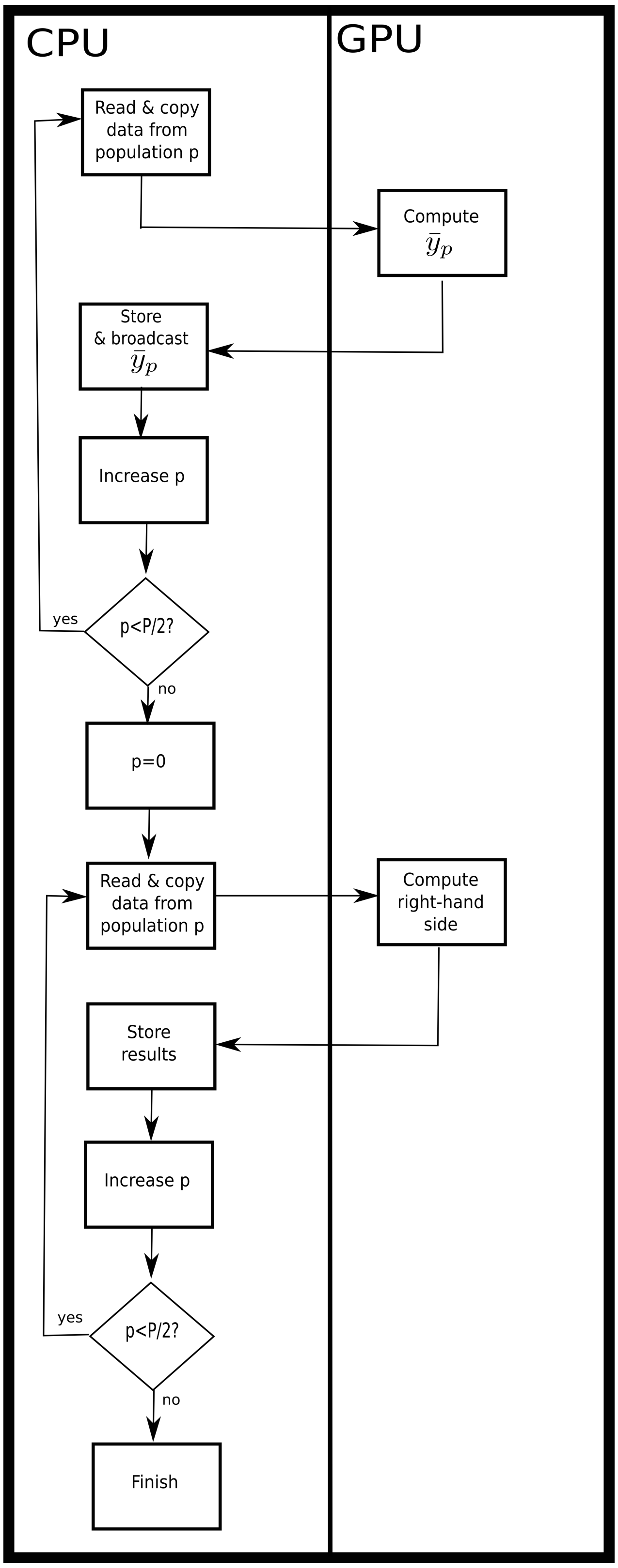


Figure 2.42: Flow diagram showing a way to simulate more populations by sequentially updating each one

have used in the previous experiments, is an extremely hard task. A non linear system of $308^3 = 29,218,112$ equations would need to be solved each time. If the classic Newton method is used, each iteration of the procedure can be obtained by solving a linear system of the same size. As the number of repetitions for the Newton method may be too large, much more computations are needed to advance one time step. For smaller systems normally the gain in time step size is larger than the loss in speed due to the larger amount of operations.

To create the linear system for each iteration of the Newton method the Jacobian matrix needs to be computed. This is a square matrix with a number of rows/columns equal to the amount of equations. In our case that would require the storage of $29,218,112 \times 29,218,112 = 853,698,068,844,544$ numbers, considering single precision floating point that would require 3,414,792.28 Gigabytes of space. Clearly, this is superior to the amount of memory in any modern computer.

One approach to solve this problem when dealing with partial differential equations is to use a sparse matrix representation ([Morton 2005]). Normally each equation that is created after discretizing a PDE with the method of lines only contains a small subset of the total unknowns of the system. This is given by the finite difference formula, for example, in a second order approximation for a 3 dimensional grid, each equation will depend only on its 12 closest neighbors (2 on each side). The Fokker-Planck equation is non-local, meaning that each point on the grid depends on all the others due to the integral term. For this reason the Jacobian matrix is a dense matrix whose storage is impossible.

Another option is to use Jacobian free methods [Knoll 2004], where this matrix is neither computed nor stored in memory but its multiplication by a vector is approximated by several computation of the right hand side of the equations. In these methods the linear system is solved by Krylov methods that only require the Jacobian for this kind of multiplication. The amount of computations of the right hand side is largely increased and depends on the convergence of the Newton method. For some cases a good preconditioner can be found that improves the speed.

We have tried to use this approach for the Fokker-Planck equation by creating an implementation in the PETSc library, which includes a parallel version of this kind of Jacobian free methods. With the simplest possible implicit integration scheme (backward Euler) and 98 processors we could not even compute one time step due to time limitations. This is due to the large computational time needed to compute the right hand side of the equations and the increased number of times this needs to be done for Jacobian free methods. When writing this thesis we didn't know of any library providing this kind of method for a multi GPU cluster.

A different approach is to use a different set of explicit methods, called Runge-Kutta-Chebyshev (RKC), that have been designed for mildly stiff problems. The formulas were originally proposed in [van Der Houwen 1983] and then extensively studied in [Verwer 1990, Abdulle 2001]. A good review of the original method and its first analysis is found in [Verwer 1996]. Its objective is to provide explicit formulas for integrating ODEs that have a stability region (area in which the eigenvalues of the Jacobian of the ODEs need to be for the system to be stable) that includes a narrow strip along the negative axis. It is common that the eigenvalues of the Jacobian of systems coming from the discretization of parabolic partial differential equations are located in this area. For these kind of problems the standard Runge-Kutta would be unstable. The better RKC method have the longest possible stripe, including in this way the biggest range of possible eigenvalues.

As both RKC and Runge-Kutta 4 methods are explicit, the formulas are similar. We changed the code to use this new method based on the implementation on [Sommeijer 1997]. This required just small modifications of the previous code and a change in some parameters. After, we tested if an increase in the size of the time step was possible. The limit for the appearence of large numerical errors was the same as before, not providing any change with the new method. This is probably due to the effect of the integral ($\bar{y}$) on the spectrum of the Jacobian, making it move away from the normal stripe around the negative axis.

Dealing with the stiffness of the Fokker-Planck equation is a hard problem as all of the attempts described before have shown. We next describe a different method that has allowed us to increase the size of the time step and the maximum number of populations. This new approach is well suited for GPU computing and opens the door for more complex algorithms like multi grid implementations.

2.4.1 Relaxation techniques

A nonlinear system needs to be solved for each time step when an implicit method is used for the integration in time of the discretized equations (instead of Runge-Kutta 4). As mentioned before, one option is to use the Newton method. This is extremely difficult in the case of the Fokker-Planck equation. A different option is to use a relaxation methods to solve directly the nonlinear system, without computing the Jacobian. For this, a rule for updating the value of each variable is created following the original equations and then it is repeated until convergence.

There are 2 common approaches for the creation of the update rule: the Jacobi or the Gauss-Seidel type iterations [Briggs 2000, Trottenberg 2001]. In a Jacobi iterations the new value for one variable is updated considering only results from the previous iterations. Instead, in a Gauss-Seidel iteration the equations are updated sequentially and for all the previous equations the already computed new values are used. For this reason, the results depend on the order in which the variables are updated. Although this last method is known to converge faster it is much more difficult to parallelize [Tritsiklis 1989].

One way to parallelize a Gauss-Seidel iteration is to exploit the structure of the dependency between the equations. Normally, on systems that come from a finite difference discretization of partial differential equations each variable will only depend on its closest neighbours. Each variable can be labelled with a color in such a way that each point has a different tag than his neighbours. For a two dimensional grid with a first order approximation two colors are enough. This is why this method is called Red Black Gauss-Seidel. Finally, the points with the same color can be updated in parallel as there is no dependency between them.

The previous approach for the parallelization of the Gauss-Seidel iteration can't be used for the Fokker-Planck equation as each point depends on all of the other and not only on its neighbors. The minimum amount of colors would be equal to the number of equations. For this reason we need to use a Jacobi type iteration. In this case, the rule is completely parallelizable if all the threads or processes may access the information of the previous iteration. This is a single instruction multiple data type of parallelization, specially well suited for vectorized machines like GPUs.

If a nonlinear system can be described as $N(u) = 0$, where u is the unknown vector, a nonlinear Jacobi relaxation for the j equation would be of the form $N(u_0^m, u_1^m, u_2^m, ..., u_{j-1}^m, u_j^{m+1}, u_{j+1}^m ...) = 0$, being u_i^m the result of the m iteration for variable i. This is a single nonlinear equation in the unknown u_j^{m+1} that can be solved by any method independently of the others. As proposed in [Trottenberg 2001] a one step Newton method can be applied to get a rule for updating iteratively the variables.

For the discretization in time we use the backward Euler scheme. The explicit integration with Runge-Kutta 4 is unstable, so an implicit method is required. Backward Euler is the simplest implicit scheme that has a larger stable area than the explicit Runge-Kutta 4. The main difference between the two is the magnitude of the error. The Euler method is of order Δt and Runge-Kutta 4 is of order Δt^4. This makes the results obtained with the new method less reliable but they will show the general behavior of the system. If, for example, we are only interested in determining which populations are spiking and which stay close to their resting potential, as in the orientation selection model, this new integration method will give us enough information.

We have first discretized the Fokker-Planck equation (2.11) in a regular grid using the formulas of equation (2.13) and (2.14). This transforms the partial differential equation into a system of ordinary differential equations $\frac{\partial p_{i,j,k}(t)}{\partial t} = f_{i,j,k}(p,t)$ where (i,j,k) are coordinates in the discretization of the three dimensional (V,w,y) space. Time is also discretized using the backward Euler method, transforming the system of ODEs into a system of nonlinear equations where the unknowns are the values of the functions $p_{i,j,k}$ at each discrete time step t'. Each equation, for the variables $p_{i,j,k}^{t'}$, is of the form:

$$p_{i,j,k}^{t'-1} - p_{i,j,k}^{t'} + \Delta t f_{i,j,k}(p_{0,0,0}^{t'-1}, ..., p_{i,j,k}^{t'-1}, ...) = 0 = F(p_{i,j,k}^{t'}, p_{0,0,0}^{t'-1}, ..., p_{i,j,k}^{t'-1}, ...)$$

Using a Jacobi method with a one step Newton iteration gives us with the following iterative rule:

$$[p_{i,j,k}^{t'}]^{m+1} = [p_{i,j,k}^{t'}]^m - \frac{F([p_{i,j,k}^{t'}]^m, p^m)}{F'([p_{i,j,k}^{t'}]^m, p^m)} \tag{2.24}$$

where $[p_{i,j,k}^{t'}]^m$ is the result of the m^th iteration for the $p_{i,j,k}^{t'}$ variable and p^m is a set with the result for all the variables at iteration m.

The Fokker-Planck equation can be divided in a set of diffusion and drift terms. When discretized all of them will become linear elements except for the ones associated to the V variable (see equation ((2.11))). The two terms that create the nonlinearity will be analyzed next, a special consideration will be given to the computation of the derivatives as this is needed for the one step Newton method included in the relaxation rule proposed.

The first term is:

$$-\frac{\partial}{\partial V}\{[V-\frac{V^3}{3}-w+I-\bar{J}(V-V_{rev})\int yP_\phi(t,V,w,y)dVdwdy]P_\phi(t,V,w,y)\}$$

The discretization of this term (using formulas in equation (2.13) and (2.14)) is:

$$\begin{aligned}
&-[V_{i-2,j,k}-\frac{(V_{i-2,j,k})^3}{3}-w_{i,j,k}+I-\bar{J}(V_{i-2,j,k}-V_{rev})Int(p)]p_{i-2,j,k}\\
&8[V_{i-1,j,k}-\frac{(V_{i-1,j,k})^3}{3}-w_{i,j,k}+I-\bar{J}(V_{i-1,j,k}-V_{rev})Int(p)]p_{i-1,j,k}\\
&-8[V_{i+1,j,k}-\frac{(V_{i+1,j,k})^3}{3}-w_{i,j,k}+I-\bar{J}(V_{i+1,j,k}-V_{rev})Int(p)]p_{i+1,j,k}\\
&+[V_{i+2,j,k}-\frac{(V_{i+2,j,k})^3}{3}-w_{i,j,k}+I-\bar{J}(V_{i+2,j,k}-V_{rev})Int(p)]p_{i+2,j,k}
\end{aligned}$$

Where $V_{i,j,k}$ is the voltage of the point at position (i,j,k) of the grid, $w_{i,j,k}$ is the value of the recovery variable at position (i,j,k) of the grid, and $Int(p)$ represent a discrete integral operator.

To apply the one step Newton we derive this equations with respect to $p_{i,j,k}$. This variable only appears inside the integral operator. This discretized version of the integral can be consider as $Int(p)=\sum_{i,j,k}W_{i,j,k}p_{i,j,k}$, where the weight $W_{i,j,k}$ depends on the integration rule (trapezoidal, Simpson, etc..). When deriving the only terms that remain are:

$$\begin{aligned}
&\bar{J}(V_{i-2,j,k}-V_{rev})W_{i,j,k}p_{i-2,j,k}-8\bar{J}(V_{i-1,j,k}-V_{rev})W_{i,j,k}p_{i-1,j,k}\\
&+8\bar{J}(V_{i+1,j,k}-V_{rev})W_{i,j,k}p_{i+1,j,k}-\bar{J}(V_{i+2,j,k}-V_{rev})W_{i,j,k}p_{i+2,j,k}
\end{aligned}$$

The second term that causes the non linearity is:

$$\frac{1}{2}\frac{\partial^2}{\partial V^2}\{[\sigma_{ext}+\sigma_j^2(V-V_{rev})^2)(\int yP_\phi(t,V,w,y)dVdwdy)^2]P_\phi(t,V,w,y)\}$$

The discretization of this term becomes (using formulas in equation (2.13) and (2.14)):

$$\frac{1}{2}[-(\sigma_{ext}+\sigma_j^2(V_{i-2,j,k}-V_{rev})^2Int(p)^2)p_{i-2,j,k}$$

$$+16(\sigma_{ext}^2 + \sigma_j^2(V_{i-1,j,k} - V_{rev})^2 Int(p)^2)p_{i-1,j,k}$$
$$-30(\sigma_{ext}^2 + \sigma_j^2(V_{i,j,k} - V_{rev})^2 Int(p)^2)p_{i,j,k}$$
$$16(\sigma_{ext}^2 + \sigma_j^2(V_{i+1,j,k} - V_{rev})^2 Int(p)^2)p_{i+1,j,k}$$
$$-(\sigma_{ext}^2 + \sigma_j^2(V_{i+2,j,k} - V_{rev})^2 Int(p)^2)p_{i+2,j,k}]$$

If we derive this with respect to $p_{i,j,k}$ we have

$$-15\sigma_{ext}^2 - \sigma_j^2(V_{i-2,j,k} - V_{rev})^2 p_{i-2,j,k} W_{i,j,k} p_{i,j,k}$$
$$+16\sigma_j^2(V_{i-1,j,k} - V_{rev})^2 p_{i-1,j,k} W_{i,j,k} p_{i,j,k}$$
$$-15\sigma_j^2(V_{i,j,k} - V_{rev})^2 3p_{i,j,k}^2 W_{i,j,k}$$
$$+16\sigma_j^2(V_{i+1,j,k} - V_{rev})^2 p_{i+1,j,k} W_{i,j,k} p_{i,j,k}$$
$$-\sigma_j^2(V_{i+2,j,k} - V_{rev})^2 p_{i+2,j,k} W_{i,j,k} p_{i,j,k}$$

This analysis shows that evaluating the derivative required for the iterative rule of equation (2.24) is not a hard computational task, as most of the elements of the discretized version of the Fokker-Planck equation are removed. The weights for computing the discretized version of the integral ($\bar{y}$) of the previous equations can be computed before the simulation and reused. The values of the neighbours are read only once as they are necesary for evaluating the right hand side of the original equations.

The main difference between the computations done for the Runge-Kutta 4 method and those for this new method is the need of the derivatives. This will only add 44N multiplications and 18N sums, a small number that can be done in parallel on the GPUs and the amount of time required for realizing them doesn't affect the total execution time. The computation of any of the steps for the Runge-Kutta 4 method should take a similar amount of time as the required for one iteration of the relaxation approach.

For each time step of the Runge-Kutta 4 method the right hand side of the equations needs to be computed four times. In the relaxation scheme the iteration rule should be repeated until convergence (or until a desired level of error is achieved). The number of repetitions for this last scheme might be much larger for one time step. This may make the new method look like an extremely slow process. This is not true as the size of the time step is much different in the two cases. For the Runge-Kutta 4 method it is limited to a very small number while due to the implicit Euler method it is much bigger for the relaxation scheme. For this reason the number of computation to advance from certain time t_1 to another time t_2 may be smaller with the iterative technique.

The number of calls to the right hand side to advance from t_1 to t_2 in the Runge-Kutta 4 method is $4 \times \frac{t_2 - t_1}{\Delta t_{RK4}}$ while for the relaxation technique is $N_{iter} \times \frac{t_2 - t_1}{\Delta t_{iter}}$. If this new approach lets us increase enough Δt and the number of iterations, N_{iter}, to reach to an acceptable error level is not too large it should be faster than the explicit method. The fact that the kind of computation of a Jacobi type iteration are extremely well suited for our hardware is a plus and should influence the difference in speed. Figure 2.43 shows a diagram of an example of this situation, where using both methods the value of the function at time T is computed. In this figure, the time step for Runge-Kutta 4 is Δt_1 and for the relaxation method is Δt_2, with $\Delta t_2 > \Delta t_1$. The number of iterations of the relaxation method in order to obtain a low error is 10. In this case, the total number of computations of the right hand side (black circles on the diagram) is smaller for the relaxation method.

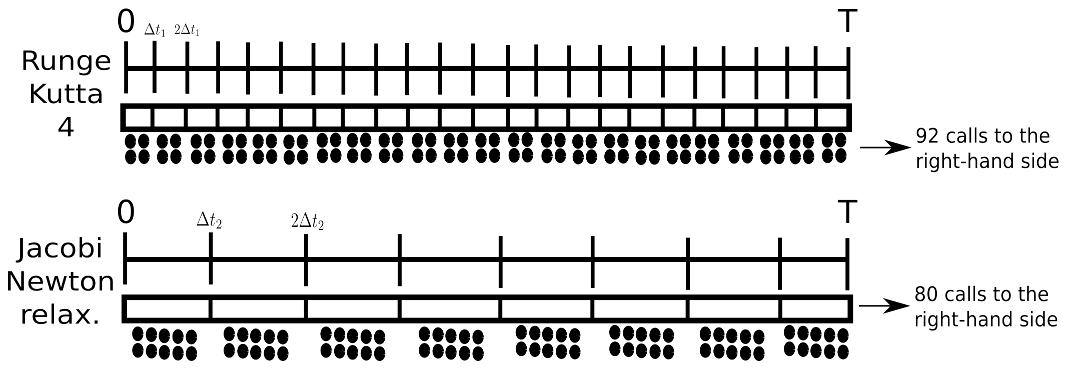


Figure 2.43: Diagram showing the amount of computations done for the Runga Kutta 4 method and for the relaxation method. For each method a time line is shown with the discrete time steps indicated in it. The rectangle below each time line represent the computations and it is divided in smaller elements which indicate each complete step of the method. The black circles below each smaller rectangle correspond to the amount of calls to the right hand side performed for that step.

Another element that affects the convergence of any relaxation technique is the initial solution. The iteration starts from a first guess and the closest this point is to the real solution the less amount of iterations that are needed to reach it. We propose to start the simulations by computing an approximation to the probability density by solving the network equations for a small number of neurons. This can be done in a Monte Carlo fashion. The larger the number of cells and executions of the Monte Carlo simulationw the better the starting point.

The solutions of this small network equations can be computed in parallel using the multiple GPUs available in our hardware. This computational power allows us to use a large number of neurons, reducing the number of necessary iterations. It is also possible to improve the approximation of the probability density by taking advantage of the propagation of chaos effect. As each neuron is an independent sample, at each time step of the simulations of a network with N neurons, it is possible to obtain N samples of the process. This reduces the number of Monte Carlo simulations, and makes it possible, if the number of neurons is large enough, to use only one.

We have created the necessary software to use this new method for the Fokker-Planck equation of the FitzHugh-Nagumo network. This new version also uses the same multi GPU hardware as before. First an initial approximation is obtained by

dt	Number of iterations	Number of neurons in the initial guess	Execution time
0.01	20	999964	51 minutes
0.01	30	999964	74 minutes
0.02	20	999964	26 minutes
0.02	30	999964	38 minutes
0.01	20	4999820	62 minutes
0.02	30	4999820	31 minutes

Table 2.7: Execution times of a complete simulation of the Fokker-Planck solver for a grid of $228 \times 228 \times 228$ using the relaxations scheme with different configurations

simulating the network with a limited amount of neurons and next this probability density is improved following the iterative procedure described before. The implementation is similar to the Runge-Kutta 4 method: the points are divided equally into the processor-GPU pairs and only the boundaries between these sub-domains are communicated through shared memory in the same machine and through an MPI message passing between machines. The only difference between the two codes is the rule applied and the number of times it is computed.

Table 2.7 shows the execution time for the new solver with different configurations for a $228 \times 228 \times 228$ grid (smaller than before) and up to time 150. The previous version with the explicit Runge-Kutta 4 method takes **87 minutes** for the same simulation with time step 0.001. The parameters that have been changed (dt, number of neurons and number of iterations) determine both the quality of the solution and the time required for obtaining it.

In all the simulations presented on table 2.7 the execution time is smaller than with the explicit Runge-Kutta method. The minimum difference is 13 minutes and the fastest one reduces the time to less than one half the previous value. This is mainly due to the larger size of the time step which on these new simulations is 10 or 20 times bigger than before. Although executing one step is longer than with Runge-Kutta 4, the total number of steps required to reach the same time is smaller.

2.4.2 Extended multi population simulations

The new faster method for obtaining the solution of the Fokker-Planck equation together with a reduction in the size of the grid allowed us to increase the maximum number of populations. With this new configuration we could simulate one different population on each card, giving a total of 14. This new implementation was used to increase the number of orientations in the model described in section 2.3.3.

As in the original model each orientation is represented by two populations, one excitatory and one inhibitory. With this new implementation we extend the number of angles from 3 to 7. The structure of the connectivity was mantained as in figure 2.34: the excitatory populations are connected with each other and to their corresponding inhibitory group while each of the inhibitory populations is connected to all the excitatory populations. The size of the weight depends on the distance between the preferred orientations, similar angles reinforce each other while different ones diminish the activity of each other.

The input for all the simulation was set to 1.0 for the population in the center and it was reduced according to the distance. The two populations next to the center received 0.7, the next 2 received 0.4 and the final 2 received 0.1. This represents an initial detection of an angle very close to the prefered orientation of the population in the center. A plot of this function is shown in figure 2.44.

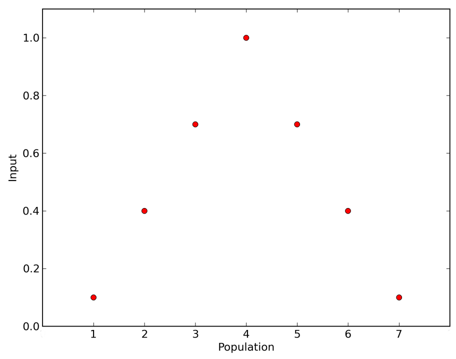


Figure 2.44: Input to the extended multi population experiment

We made several simulations keeping all the parameters the same as in the experiments presented in 2.3.3 and external noise 0.45 (value for which we found a stationary solution). The difference was in the level of inhibition, a critical value for the sharpening effect expected from the system. In each new experiment the weights of the inhibitory connections were duplicated. The amount of iterations was set to 30 and number of neurons in the initial guess to 99,964. Figures 2.45, 2.46 and 2.47 show the stationary solution of the excitatory populations. Only four of them are shown as in all the experiments the probability density of populations at the same distance of the center was equal.

In the first experiment (see figure 2.45) all of the populations are spiking as they present some mass around the limit cycle. This means that a percentage of the neurons are following this trajectory, i.e., they are emiting action potentials. The high peak at a low voltage for the population furthest away from the center indicates that the majority of the cells are staying very close to their resting potential. This peak is diminishing when approaching the center and finally it is almost completely removed and replaced by another one but at a high voltage. More neurons are spiking for populations with higher input, this is a good representation of the original angle selection but has no sharpening of the original values. The results are similar as for the first experiments of section 2.3.3 shown on figure 2.35.

In the second experiment (see figure 2.46) the activity of the first population is completely removed as all the mass is close to a negative voltage. This is an effect

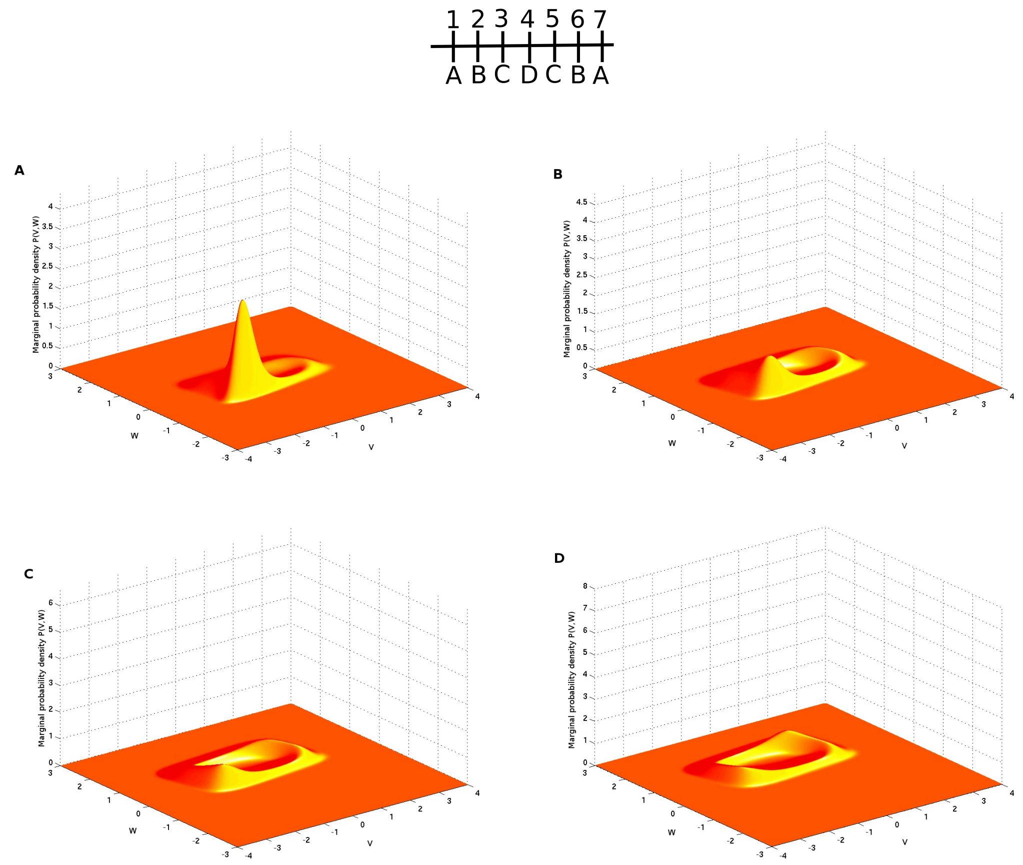


Figure 2.45: Stationary solution of the excitatory populations for the experiment with 14 populations and smaller level of inhibition. A: populations furthest away from the correct orientation. B: populations next to the furthest away from the correct orientation. C: populations neighboring the correct orientation. D: population with the correct orientation. See map at the top of the figure for details on the position of the populations. Movies for this simulation are available in the web site

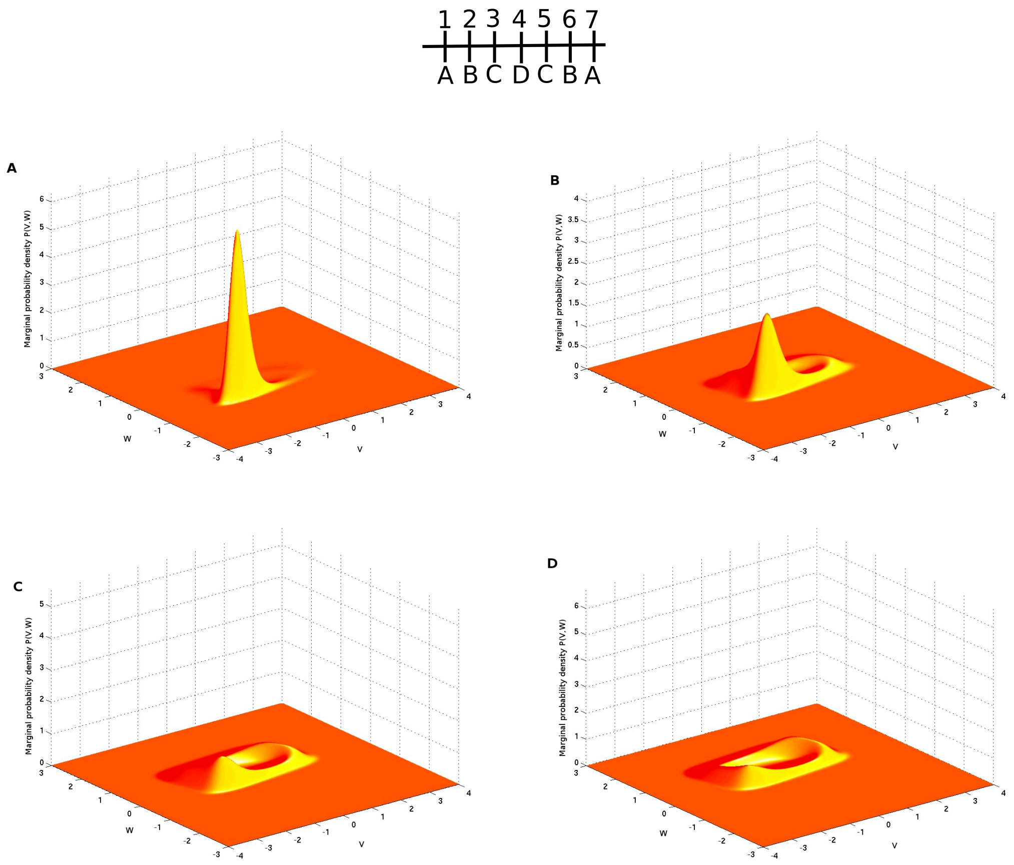


Figure 2.46: Stationary solution of the excitatory populations for the experiment with 14 populations and medium level of inhibition. A: populations furthest away from the correct orientation. B: populations next to the furthest away from the correct orientation. C: populations neighboring the correct orientation. D: population with the correct orientation. See map at the top of the figure for details on the position of the populations. Movies for this simulation are available in the web site

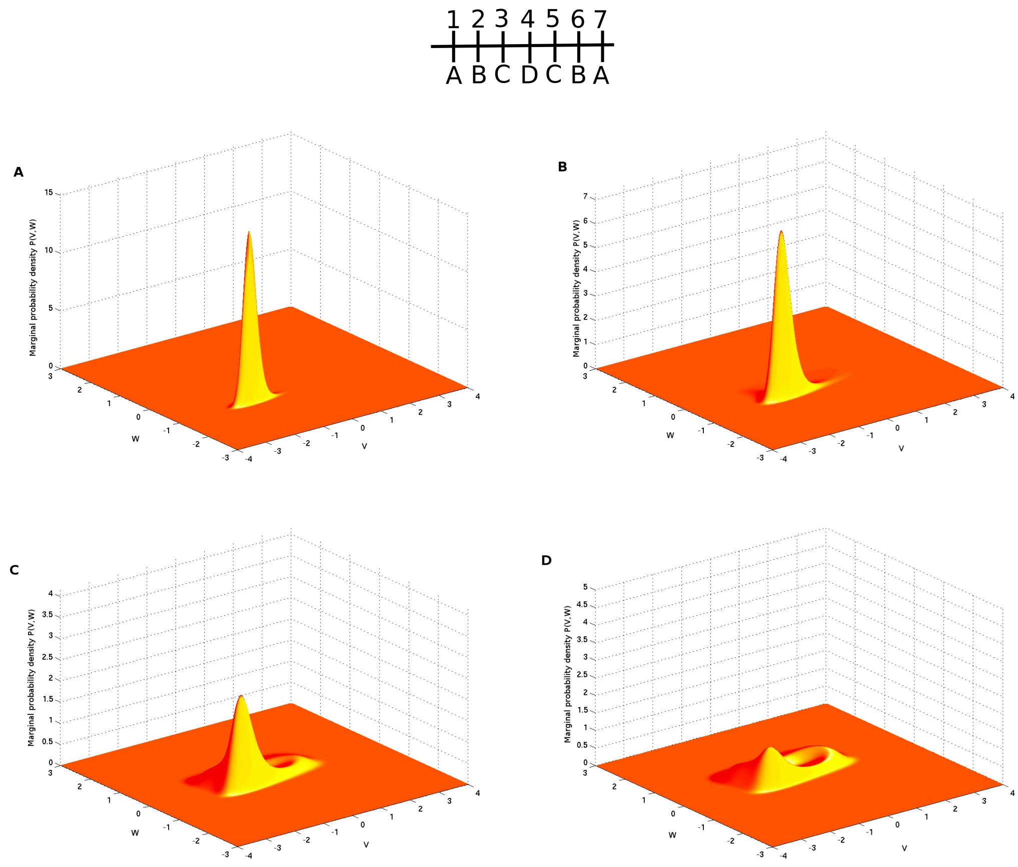


Figure 2.47: Stationary solution of the excitatory populations for the experiment with 14 populations and high level of inhibition. A: populations furthest away from the correct orientation. B: populations next to the furthest away from the correct orientation. C: populations neighboring the correct orientation. D: population with the correct orientation. See map at the top of the figure for details on the position of the populations. Movies for this simulation are available in the web site

of the increase in inhibition. All the other populations are spiking as they feature probability mass around the limit cycle. The one in the center still has a larger level of activity than the others as the peak at low voltage is very small. The sharpening effect is small as only one orientation was eliminated.

In the third experiment (see figure 2.47) the sharpening effect is much more clear as the 2 populations furthest away from the center show no activity and the one next to the center has a very low amount of neurons spiking. The populations that are neighbours to the one in the center have a high peak at the resting potential and a small amount of mass around the limit cycle. This means that a very low proportion of the neurons are spiking. This is extremely different to the solution found for the population in the middle where although there is a peak at a negative potential, the majority of the mass is around the cycle. This is the kind of behavior we were expecting of the system as the difference between the activity at the selected orientation and the others is larger than in the input.

In a final experiment, we left the weights as in the experiment of figure 2.47 but changed the input function. We now moved the input orientation from the center position to one of its neighbours. A plot of this function is presented in figure 2.48. The results of the experiment are presented in figure 2.49. The stationary solutions found in this experiment are similar to those presented in figure 2.47. The populations far from the correct orientation stay close to the resting potential, the one just next to the correct orientation has a small amount of mass around the limit cycle, and the one with the correct preference has the majority of its mass around the cycle.

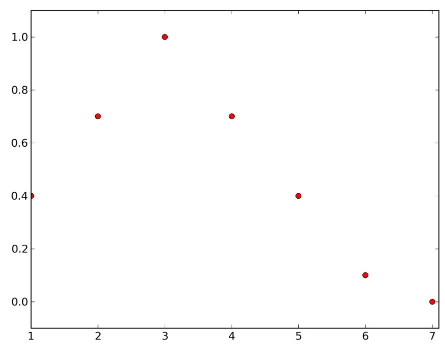


Figure 2.48: Input to the extended multi population experiment with shifted orientation

2.4.3 Discussion

The new method proposed for solving numerically the Fokker-Planck equation provides a great advantage, it is possible to tune its behavior depending on the available computational time. The slowest possible algorithm will provide the better results

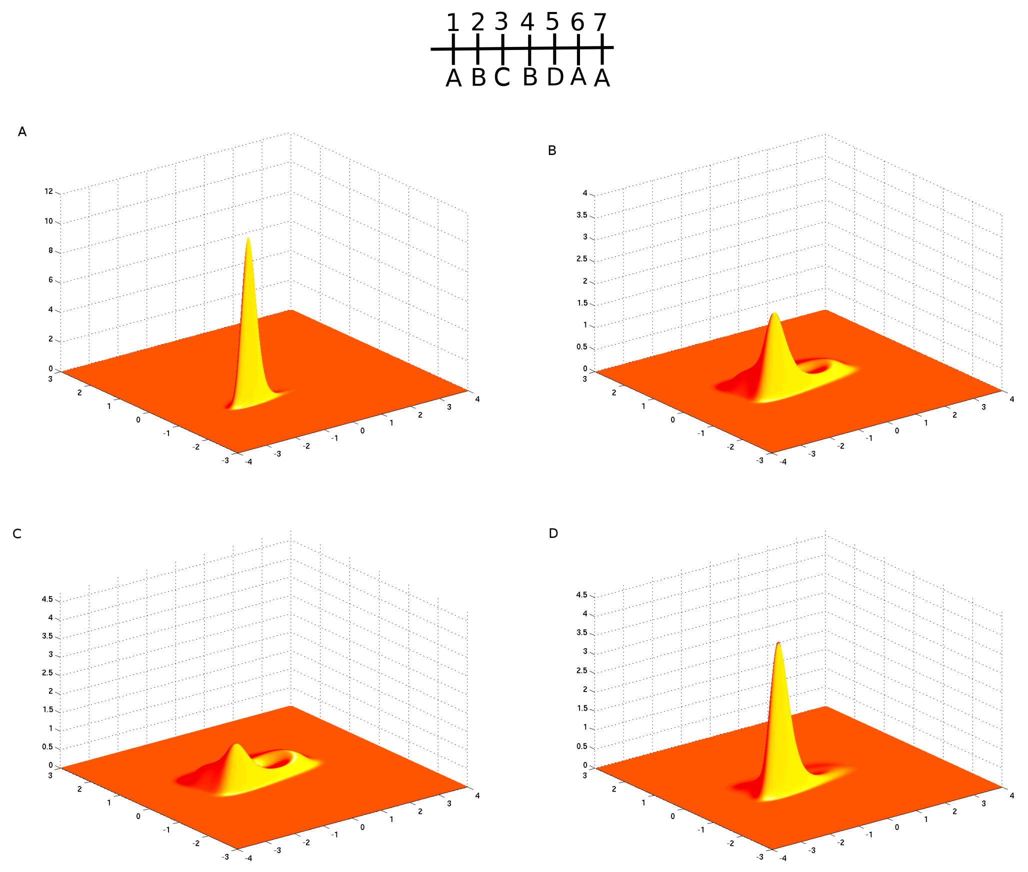


Figure 2.49: Stationary solutions of the experiment with the input function presented in figure 2.48.

(or at least will produce the smaller error bound). Depending on the objective of the simulation the number of iterations, the size of the time step or the number of neurons for the initial approximation can be changed. If the interest is just in the general dynamics a very fast implementation can be obtained. This is impossible with the Runge-Kutta implementation where the main parameter that determines the speed of the simulation, the size of the time step, needs to be extremely low.

Although it is known that the Jacobi-Newton type of updating rule we are using are slow to converge (see [Briggs 2000, Trottenberg 2001]) we could obtain a solver that is faster than the Runge-Kutta method. This is mainly due to the creation of a good initial solution and to the computational power provided by the GPUs. The possibility to obtain a first approximation of the probability density by a fast simulation of the network reduced the number of necessary iterations and allowed us to exploit the propagation of chaos effect in favor of a faster implementation. The GPUs allow us to execute one iteration extremely fast, with a speed probably only comparable to extremely large high performance computing solutions.

Before, we provided two different simulation techniques for studying numerically the kind of noisy neural network presented in this thesis. The first, useful for small networks, is to do a Monte Carlo simulation to generate samples and then create an approximation of the probability density. The second one, useful for larger networks, is to solve directly the Fokker-Planck equation. Due to the large grid size required by the Fokker-Planck equation (to avoid negative values) this approach is useful as a simulation tool only if the number of neurons in the network is large enough. The new method based on relaxation techniques is an intermediate point between the two previous options. It combines the network simulation with the solving of the Fokker-Planck equation and provides a solution that may be useful for medium to big size networks.

A possible extension to the relaxation scheme is to use it in a multigrid method [Briggs 2000, Trottenberg 2001]. In this kind of approach the solution of the equation in a smaller grid is used recursively to improve the solution obtained through iteratives techniques in a larger grid. As part of the iterations are made for a smaller amount of points the algorithm is faster than applying directly the relaxation. This combined with an adaptive mesh where the grid on different pieces of the domain may have dissimilar amount of points depending on the solution, may improve greatly the results presented in this section. As the majority of the mass is distributed along the limit cycle an adaptive algorithm would use a fine grid around it and a coarse grid outside. Several libraries have been created that provide this kind of algorithms for partial differential equations, like uG ([Bastian 1994, Bastian 1997]) or ALUGrid ([Dedner 2004]). Providing an implementation with the use of this libraries is out of the focus of this thesis and it is left for future work. A diagram showing how multigrid methods work is shown in figure 2.50.

We did some experiments for improving the iterative method proposed, in a multigrid fashion and using the GPU cluster. We created the necessary code to solve the Fokker-Planck equation with different grid sizes and then combine the

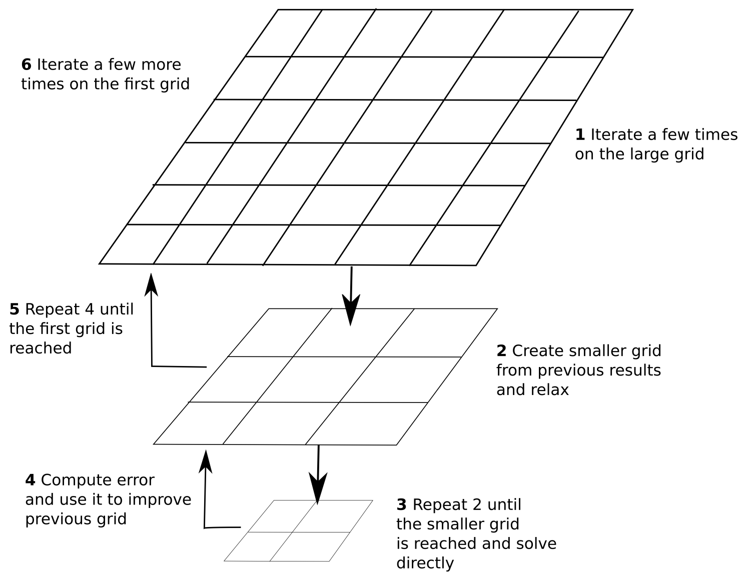

6 Iterate a few more times on the first grid
1 Iterate a few times on the large grid
5 Repeat 4 until the first grid is reached
2 Create smaller grid from previous results and relax
4 Compute error and use it to improve previous grid
3 Repeat 2 until the smaller grid is reached and solve directly

Figure 2.50: Diagram showing how multigrid methods work

results. The experiments with this implementation showed that solving in a GPU a grid with a reduced amount of points will not decrease the time by the same factor. In fact, for smaller grids, the diminution of time is almost none. This ends up making the code even slower than the version that just relaxes directly the largest discretization. The fact that this difference in execution time is smaller than expected is probably due to hardware limitations. For this reason we believe that if these experiments were repeated on a different hardware architecture they could be successful.

The numerical experiments with 14 populations reinforce what was found previously with just 6. The system is able to enhance the original angle selection and in this case it even destroys all the activity of the population with orientations dissimilar with the input. As previously, the level of inhibition is the key element to determine the size of the sharpening effect. On the experiments all the populations arrived to a different stationary solution that depends on the distance to the input angle. Probably, if more orientation were added to the model the same effect would be seen and only the ones representing angles similar to the peak in the input function would survive.

The results also show how a population of neurons can be defined as a basic computational unit, instead of an isolated single neuron. Measurements in the visual area show how large groups of neurons share common properties and are strongly connected. This may be a sign of the existence of networks with enough elements to be in the mean field limit and behave as the ones used for the simulations presented in this work. New measurements could be made to show the existence of a stationary distribution although the activity of a huge amount of near neurons should be recorded for a static input.

Chapter 3

Numerical simulation of neural field models of the primary visual cortex

Contents

3.1 A model without feature based connectivity

3.1.1 Motivation

In [Veltz 2011] the author proposed a new neural field model of the primary visual cortex. The main difference with previous approaches was the use of a connectivity that doesn't depend on any feature. This is supported by some biological data, like the one presented in [Bosking 1997] where a columnar pinwheel structure is shown to be present in V1. These results are confirmed by the work described in [Lund 2003], where the conclusion was that the patchy lateral connections are the only clearly identifiable example of an anatomical column. In other rate models of V1, like those in [Ben-Yishai 1995, Bressloff 2001b, Chossat 2009], the connectivity between elements depends on the contour orientation, a feature which is explicitely represented in the equation, together with the position.

Another difference between this model and others in the literature is that the existence of a continuum of pinwheels covering the cortex is not assumed (see

[Bressloff 2001b, Bressloff 2002a]). This is not realistic, as the number of pinwheels in any orientation map is limited and because the linear zones between pinwheels must be neglected if a continuum is assumed. Even though they do not use a realistic orientation preference structure these models have been shown to account for several interesting phenomena. The Veltz model doesn't require such an assumption and can represent an orientation map with linear zones and a limited amount of pinwheels.

Veltz provided an analytical study of his model based on his work with the Ring Model of Orientation. Due to limitations in computational power and time he was not able to perform numerical simulations to test his theoretical results or to guide the analysis with the use of numerical methods when analytical methods became too complicated.

In this chapter we extend the work done in [Veltz 2011] by providing simulations of his model and numerical tests for several hypotheses he proposed. First, we describe the model and second we present a set of algorithms and software packages to solve the model equations on a GPU architecture. Finally, the results of several numerical experiments will be detailed, one for each of the predictions in the original document.

3.1.2 Description of the model

The model consists of a neural field equation which depends only on the cortex position x, i.e. there is no explicit representation of the edge orientation at x.

$$\tau \frac{d}{dt} V(x,t) = -V(x,t) + \int_{\Omega} J(x,y) S[\sigma V(y,t)] dy + I_{ext}(x), \tag{3.1}$$

where Ω is a square two-dimensional piece of the cortex and $S(x) = \frac{1}{1+e^{(-x+T)}}$. The boundary conditions are periodic, i.e. Ω is assumed to be surrounded by other pieces of the cortex with the exact same values of V (the space has a torus shape).

The weight function J is composed of a local connectivity and a modulatory lateral connectivity. The first one is homogeneous, i.e., it depends only on the distance between the points. The second one spreads in the direction of the preferred orientation following biological constrains.

$$J(x,y) = J_{loc}(x-y) + \varepsilon_{LR} J_{LR}(x,y). \tag{3.2}$$

The local connectivity is modeled as the following difference of Gaussians:

$$J_{loc}(x) = a e^{-\frac{||x||^2}{2\sigma_{loc}^2}} - e^{-\frac{||x||^2}{4\sigma_{loc}^2}}. \tag{3.3}$$

Most of the original analysis deals only with local connections and hence the first groups of numerical experiment described below will also only consider this kind of connectivity.

The input function depends on an orientation map, that could be obtained in biological experiments with optical imaging, for example. The map is an assignment of an angle between $-\frac{\pi}{2}$ and $\frac{\pi}{2}$ to each position in the cortex. When there is a single pinwheel the value of the orientation depends only on the polar angle from the center of the structure. In this case, given an afferent stimulus angle, written θ_{aff} we have that:

$$I_{ext}(x) = \varepsilon[1 + \beta \cos(2\theta(x) - 2\theta_{aff})], \tag{3.4}$$

where ε is the contrast, β is a small number representing the anisotropy of the LGN input, $\theta(x)$ is the preferred orientation of x and θ_{aff} is the stimulus angle. When there is more than one pinwheel this function may be used with different parameters or stimulus angle at each one.

It is assumed that the pinwheels are distributed in a square lattice and a procedure for building the map is provided. First, build a $\pi \times \pi$ square hypercolumn, consisting of one pinwheel, like the one presented in figure 3.1 top. Then a rectangle of size $2\pi \times \pi$ is generated by reflecting along the vertical axis. Reflecting again this along the horizontal axis yields a $2\pi \times 2\pi$ square which is used as a basic tile for building a larger map. Finally, this tile can be repeated as many times as needed to generate a square orientation map such as the one on the bottom of figure 3.1.

The lateral connections follow the same biological constrains as in section 1.2.4.2. Populations connect only if they have similar preferred orientations, creating a patchy connectivity. This is modeled by a Gaussian function, $G_{\sigma_\theta}(\theta(x) - \theta(y))$. It must also follow the direction of the preferred orientation. This can be modeled by the term $J_0(\chi, R_{-2\theta(x_0)}(x_0 - y))$ where $J_0(\chi, x) = e^{-[(1-\chi)^2 x_1^2 + x_2^2]/2\sigma_{LR}^2}$ and $R_{2\theta(x)}$ is the counter-clockwise rotation of angle $2\theta(x)$. If $\chi = 0$, then there is no anisotropy whereas for $\chi \in (0, 1)$, this connectivity presents an anisotropy along the preferred direction. Finally, the lateral connectivity can be described the following equation taken from [Veltz 2011]:

$$J_{LR}(x, y) = J_0(\chi, R_{-2\theta(x)}(x - y)) G_{\sigma_\theta}(\theta(x) - \theta(y)). \tag{3.5}$$

Figure 3.2 shows an example of the lateral weight function for a point close to the center of the same pinwheel grid as in figure 3.1 bottom and with orientation $\frac{\pi}{4}$. The value of χ is 0.8 and $\sigma_{LR} = 0.15$.

3.1.3 Implementation

3.1.3.1 Local connectivity

If long range connections are removed, the integral term in the right hand side of equation 3.1 is a convolution for which a Fast Fourier Transform (FFT) algorithm can be used to reduce the complexity. Together with CUDA, nVidia provides a set of libraries for solving common problems on its GPUs, one of them, called cuFFT, being designed to apply FFT in parallel, called cuFFT. We have used cuFFT to simulate the model proposed in this section with only local connections.

The software starts by computing a local weight matrix and then its FFT on the GPU. The results of this operation remain in the card and will be used during the

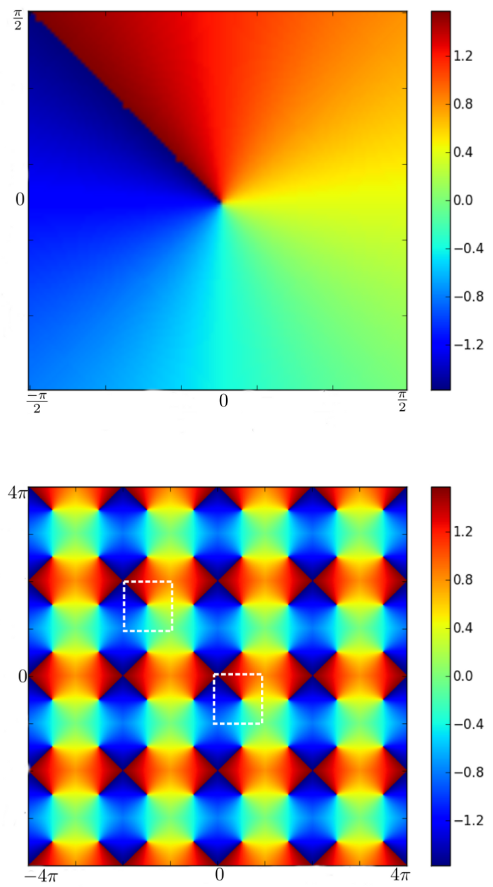


Figure 3.1: Top: distribution of orientation preferences in one pinwheel. Bottom: orientation map composed of a square grid of 8×8 pinwheels. The white square indicate two example pinwheels.

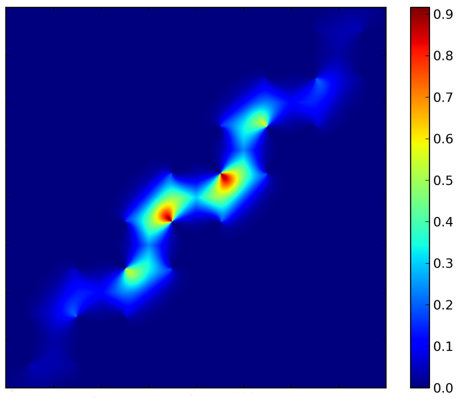


Figure 3.2: Lateral weight for a point in the center of the grid. See text for more details.

whole simulation. Then, for each of the four computations of the right hand-side required by Runge-Kutta 4, the sigmoid for each point is computed and then the FFT of this matrix is obtained. The final results of the convolution operator are obtained by computing the reverse transform of the multiplication of the 2 FFTs evaluated previously. After the value of the convolution is known for each point a closing process updates the values following the form of the right-hand side. All of these operations are executed in parallel. A flow diagram of the process is shown in figure 3.3.

There are two main advantages of this approach: the use of a highly optimized library and the small amount of resources required by the simulation. The library for computing FFTs in the GPU is designed by the same company who created the hardware, it is extremely fast and provides a large set of optimization routines that depends on the structure of the data. This process runs on just one GPU, using a smaller amount of resources than, for example, the Fokker-Planck simulations where the complete cluster was required for one simulation. This advantage proved to be advantageous when performing a numerical parameter search, where different configurations can be tested in parallel across the different cards.

As only one card is required there is no need to communicate between processors which reduced the amount of time used for copying memory. For this problem all the results are always kept in the GPU unless the values for the time step must be saved to study the evolution of the solution through time. In fact, if only the stationary solution is required no copy from GPU to CPU memory is done until the end of the simulation.

One complete simulation of the model for a 800×800 grid, 3,000 time steps (much more than is normally required for convergence) and saving the state of the network every 200 steps takes 49 seconds (average across several executions). This is

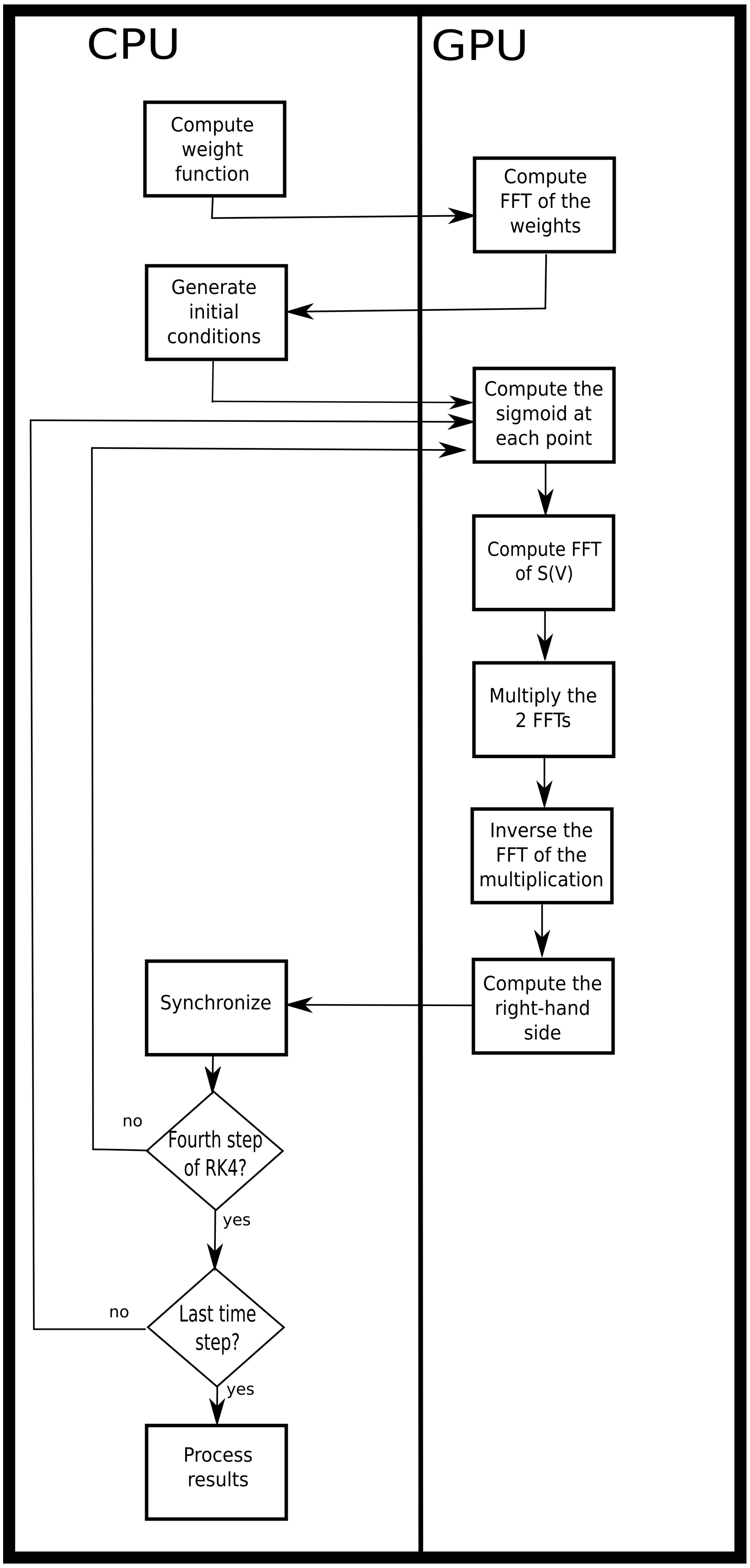

CPU
GPU
Compute weight function
Compute FFT of the weights
Generate initial conditions
Compute the sigmoid at each point
Compute FFT of S(V)
Multiply the 2 FFTs
Inverse the FFT of the multiplication
Compute the right-hand side
Synchronize
no
Fourth step of RK4?
yes
no
Last time step?
yes
Process results

Figure 3.3: Flow diagram of a simulation using FFT.

much faster than any other of the simulations presented in this thesis. The complete cluster allows us to perform 14 simulations with different configurations in around 49 seconds. This is only possible due to the fast parallel computations of the FFTs provided by cuFFT.

The experiments we have done with FFTs for solving this equation indicate a requirement for high precision. When we performed the simulations with floating point numbers in order to improve speed, errors of the order of 10^{-5} that changed the behavior of the system completely. The errors were measured by doing a simulation with no input and with a constant initial condition. In this case, the network should always converge to a constant solution. This doesn't happen unless double point precision is used and the differences are a measure of the error. The very small error induced by the low precision is spread due to the effect of the nonlinearity and makes the network converge to a completely different solution.

3.1.3.2 Lateral connectivity

The computation of the lateral connectivity is harder as it is expanded in the direction of the preferred orientation. This anisotropy prevents us from using FFT for both connectivities. Also, as each point requires a different weight matrix, coalesced memory access (consecutive threads access consecutive memory positions) is difficult to obtain. We have designed two different solutions for solving this problem that are now described.

The first solution uses another library provided by nVidia designed for operating on sparse matrices, called cuSparse. This code provides optimized routines for multipliying sparse matrices with dense or sparse vectors in the GPU. As the lateral connectivity is patchy due to the restriction in the orientation difference (G_{σ_θ} in equation 3.5) most of the weights are 0. The effect of these connections can be obtained by multiplying a dense vector, the result from evaluating the sigmoid at all the points, with a large sparse matrix (number of points^2) that contains the weights between all pairs of elements. The size of this matrix is reduced as only the non-zero elements are stored this can fit in the small amount of GPU memory for small discretizations. Figure 3.4 shows a flow diagram of this process

The second solution uses all the resources in the cluster to compute the weight function and the multiplications required by the lateral connectivity directly. This doesn't have the memory restriction from the previous implementation but requires the use of all the GPUs as more operations are necessary.

To do this last kind of computation fast enough we first set the block size equal to the amount of points in one row of the grid. Then, for each computation of the right-hand side, each thread will iterate over all the elements close to its position in order to obtain the value of the integral. This loop is done by first fixing the row and then iterating over all the points in it. This allows us to use the block shared memory (see figure 1.17) to obtain a fast access to the data. At the beginning each thread will copy one value of the input and one angle from global to shared memory. These values are required by at least one thread during the second loop. A

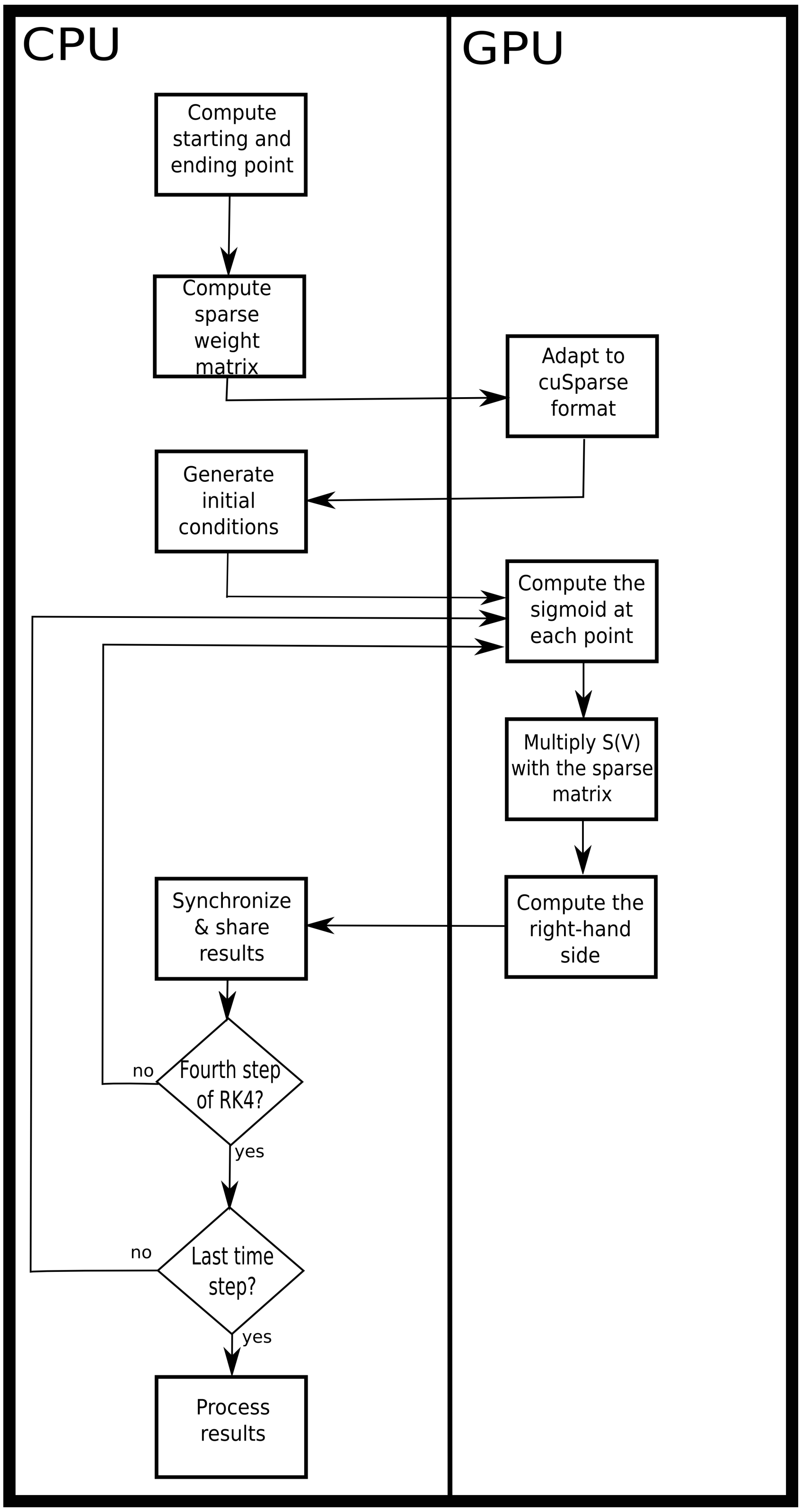

CPU
GPU
Compute starting and ending point
Compute sparse weight matrix
Adapt to cuSparse format
Generate initial conditions
Compute the sigmoid at each point
Multiply S(V) with the sparse matrix
Compute the right-hand side
Synchronize & share results
Fourth step of RK4?
no
yes
Last time step?
no
yes
Process results

Figure 3.4: Flow diagram of a simulation using a sparse matrix representation.

pseudo code explanation of the algorithm is provided below . In the code svv-local is a continous array of double precision numbers in shared memory that stores the evaluation of the sigmoid at each point of the current row and po_map_local is a similar structure that stores the orientation preferences of the points in the current row. Synchronizing the block means that all the threads in it will wait until all other threads have reached the same step of the code.

Algorithm 1 Computation of lateral connectivity with shared memory

```
for each possible row in the neighborhood do
  svv-local[thread id] ← read the value of S(input) of one element in the row
  po_map_local[thread id] ← read the angle for one element in the row
  synchronize the block
  for each point in the row and in the neighborhood do
    compute lateral weight using po-map-local for the Gaussian
    multiply the corresponding value of svv-local with the computed weight
    accumulate the value to obtain the final integral
```

This technique reduces the amount of access to global memory as most of the elements that are copied are required by more than one thread. This is because each of them should compute an integral, corresponding to the effect of the connectivity, which includes all the neighbours. As all the elements of the block belong to the same row, their neighbourhood is similar. Accessing shared memory during the second loop is several orders of magnitude faster than obtaining the same value from global memory. Although this code includes a synchronization instruction which is slow, the gain in speed due to the different memory architectures still makes the simulation faster with this technique. This same technique can be used to copy the local weights for one iteration.

The time required for one simulation is similar for both solutions, the main difference is the amount of resources used. For a 200x200 grid, 1200 time steps and saving every 20, the mean execution time for the sparse matrix version is 1 minute and 50 seconds while, using 14 GPUs with the other method, the execution time is 2 minutes and 20 seconds. Although the sparse matrix is faster in this example, the maximum grid size is limited by the amount of memory available in the GPU. A comparisson of both methods is presented in table 3.1

	Sparse matrix	**Direct computation**
Number of GPUs	1	14
Time	1 min 50 sec	2 min 20 sec
Memory limit	yes	no
Maximum grid size	yes	no

Table 3.1: Table comparing the two methods for computing the lateral connectivity

3.1.4 Numerical results

3.1.4.1 Only local connectivity

When no stimulus is input to the system ($I_{ext} = 0$) it always features a constant stable state. A bifurcation analysis done by Veltz showed that by increasing the nonlinear gain, σ, a different, tuned, spontaneous state may appear. This increase may be caused by drugs or other pharmacological substances having the effect of modifying the properties of the neuron population.

The author has found two different stable solutions that may appear when no input is presented to the network and the nonlinear gain is sufficiently large. The first one is a pattern of spots of activity and the second one is made of stripes. Both solutions, the spots and the stripes, are different and mutually exclusive. Which of them is selected by the system depends on the local weight configuration.

We have run a series of simulations to confirm the appearance of these two patterns of spontaneous activity. For a first group of experiments we selected a configuration that should produce a spots solution ($a = 1.854$, $\sigma_{loc} = 1.24$ and $T = 0.1$) and for a second group we choose one that should generate stripes ($a = 1.96$, $\sigma_{loc} = 1.19$ and $T = 0.1$). Uniform random initial conditions were created in each case. The value of the nonlinear gain was slowly increased until a solution different from constant activity was found. Figure 3.5 shows the two patterns found.

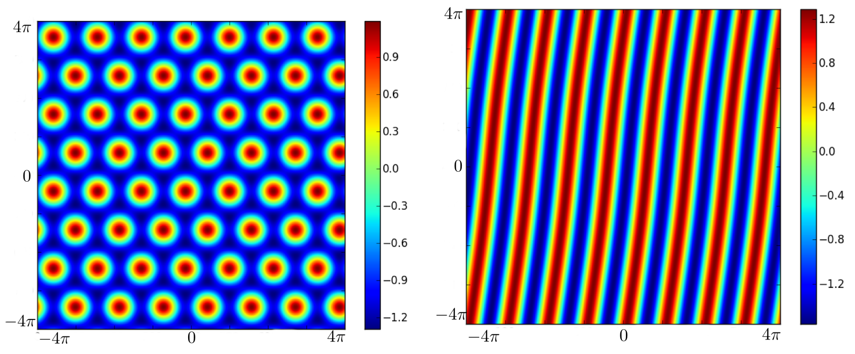


Figure 3.5: Solutions found when no input is present to the system. The sigmoid threshold, T, in both experiments is 0.1. The spot solution was obtained with $\sigma_{loc} = 1.24$ and the stripes with $\sigma_{loc} = 1.19$

The nonlinear gain, σ at which we found the first bifurcation point is close to 1.07. As recommended by Veltz in his thesis, in order to obtain the correct behavior of the system the value of σ should be chosen slightly below this point. In all the following simulations the value of σ is fixed and equal to 0.95

We did a second set of experiments to determine under which conditions the angle selected by the system is correct (i.e. the θ_{aff} in (3.4)). The predictions of the author indicate that there is at least one threshold for the contrast value (ε in (3.4)) beyond which the system produces the correct behavior. In order to check this prediction we ran simulations for different contrast values and different input orientations, using the same local weights that generated the previous spot pattern. The input was the same for all the pinwheels (full field grating). For this, all the GPUs in the cluster were used as each one ran experiments for a different set of possible configurations. The selected angle was determined by the argument of $\int V(x)e^{2i\theta(x)}dx$ and the quality of the decision by the absolute value of the same number. When computing this integral each point in the grid is considered a vector whose argument is equal to its prefered orientation and its norm the voltage. The resulting vector after adding all of them will have an argument similar to the one of the points with the higher norm (voltage) as they have a higher influence. Figure 3.6 presents the results.

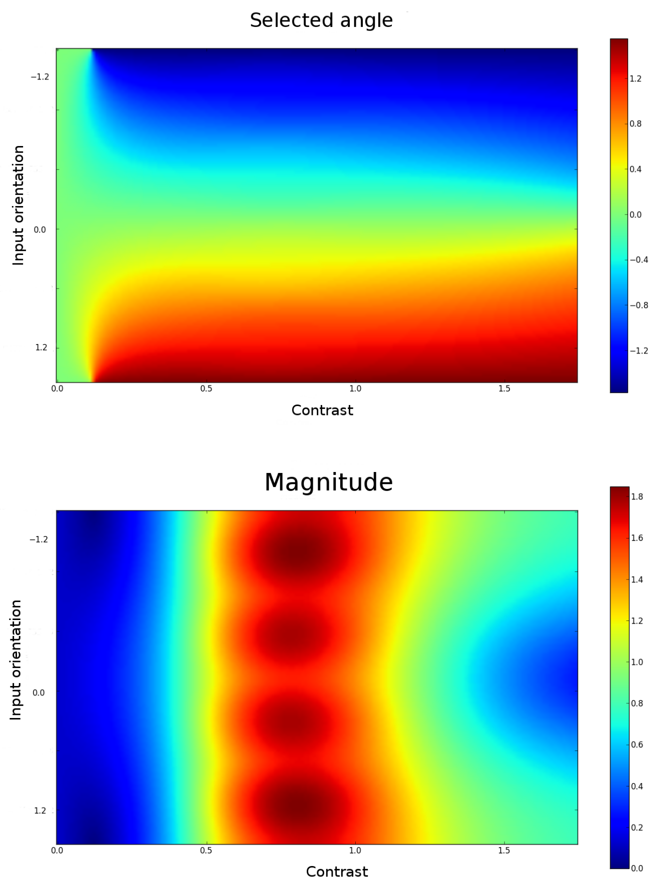


Figure 3.6: Selected angle and modulus for different configurations

The results show that for low contrast the selected angle is always very close to 0 and from a certain threshold near 0.1 the selected angle is similar to the input orientation. The modulus for this first range of contrast values is also very small indicating that the system is not really sure of the selection (or that there is no selection at all). The best behavior of the system is obtained for contrast values between 0.5 and 1.3 where the absolute value is maximal. After this contrast level the network starts to saturate and the difference in activity for the different orientations is not as large as before.

Figure 3.7 shows a plot where the horizontal axis is the orientation preference and the vertical the voltage. Each green circle represent the final solution of one point in a given pinwheel when $\theta_{aff} = 0$. The plot shows that different points with the same prefered orientation present different voltage values, but the range for the ones close to the input orientation is higher. This difference depends on the distance to the pinwheel center. Points that are furthest away from the center will connect with similar orientations while the ones that are close to the center will interact with a greater variety of angles.

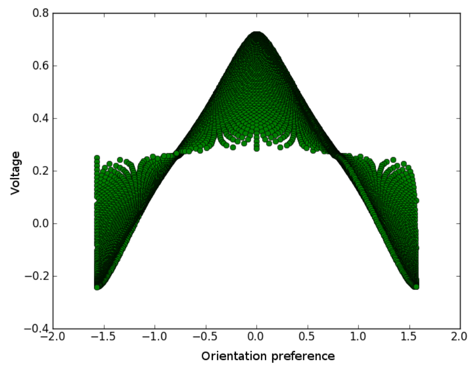
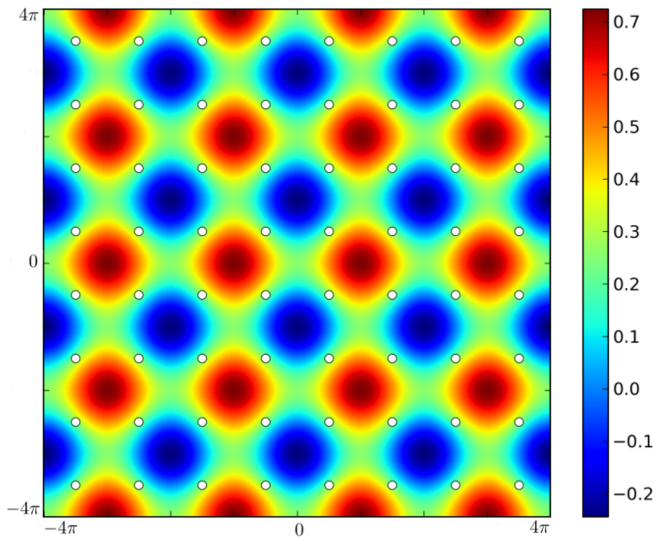


Figure 3.7: Voltages of a pinwheel for a simulation with contrast 1.0 and input angle 0.

The solution for each θ_{aff} and contrast was the same for all the pinwheels. The final solution is a spot or stripe pattern depending on the input orientation which generates a peak at different positions of each pinwheel. The final patterns indicate the structure of the orientation map, having the same organization. This is shown in figure 3.8 where the solution for an input orientation of $\frac{\pi}{4}$ and $\frac{\pi}{8}$ are shown. Each pinwheel is composed of 100×100 points and in all of them the higher values are oriented to the proper line creating a spot or stripe pattern.

In a second set of experiments the input was changed and localized to the central pinwheels. The objective of this experiments was to test Veltz hypothesis that at small contrast the system should feature a non localized solution, similar to the full field grating case, but when presented with a localized input. This is in contradiction to the recent experiments presented in [Chavane 2011]. The input for contrast 1.0 is presented in figure 3.9 and the results for several contrast values are presented in figure 3.10.

The simulations show that the original hypothesis by Veltz was incorrect and that even for low contrast the output is localized. The results are similar to the



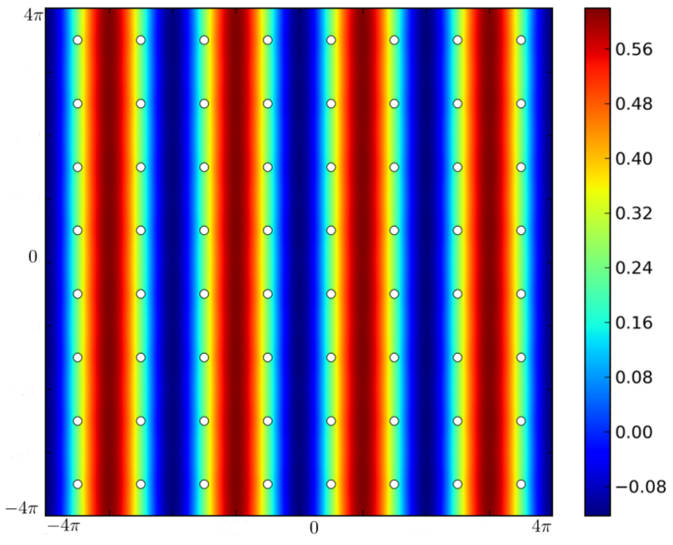


Figure 3.8: Top: final state of the network with contrast 1.0 and input orientation $\frac{\pi}{4}$. Bottom: same with input orientation $\frac{\pi}{8}$

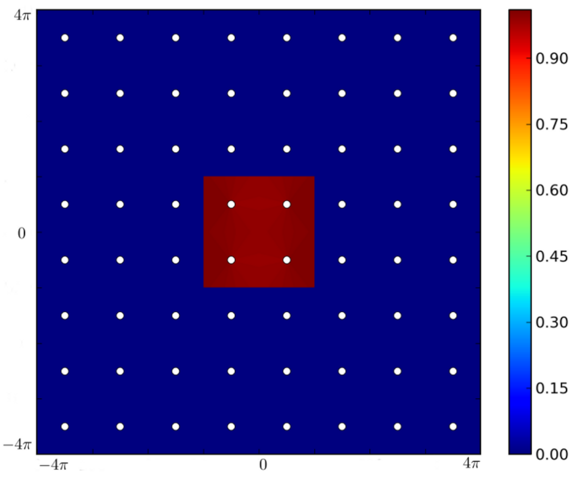


Figure 3.9: Input for the localized experiments

findings by Chavane et. al. in biological experiments. Through voltage-sensitive dye imaging the authors showed how the spread of activity in V1 is independent of the input orientation and that it doesn't cover the complete field. This is in perfect agreement with the simulations where independently of the contrast the output was always localized. So, even though Veltz hypothesis was wrong the model still agrees with biological facts.

3.1.4.2 Lateral and local connectivity

A final set of experiments was designed to test the effects of lateral connectivity. For this either the sparse matrix implementation or the one that uses the complete cluster were employed. Previous tests had indicated that the results of both version were the same. The main prediction put forward by Veltz and tested with these simulations is that long range connectivity improves the response and that this is further increased by the anisotropy, χ, for both local and full field stimuli.

The experiments consist in setting a configuration (local connectivity, contrast, input orientation and ε_{LR}) and then simulating the network with different values of χ. For each of these the modulus of $\int V(x)e^{2i\theta(x)}dx$ was measured. If the hypothesis was correct the modulus should increase when χ is increased. The results are presented in figure 3.11.

All the plots in figure 3.11 show that the modulus of the final vector increases with χ, in agreement with Veltz predictions. In fact, the shape of the curve is very similar to the one computed in his thesis using the center manifold theorem. The simulations also show that the increase is higher when the lateral connectivity in increased.

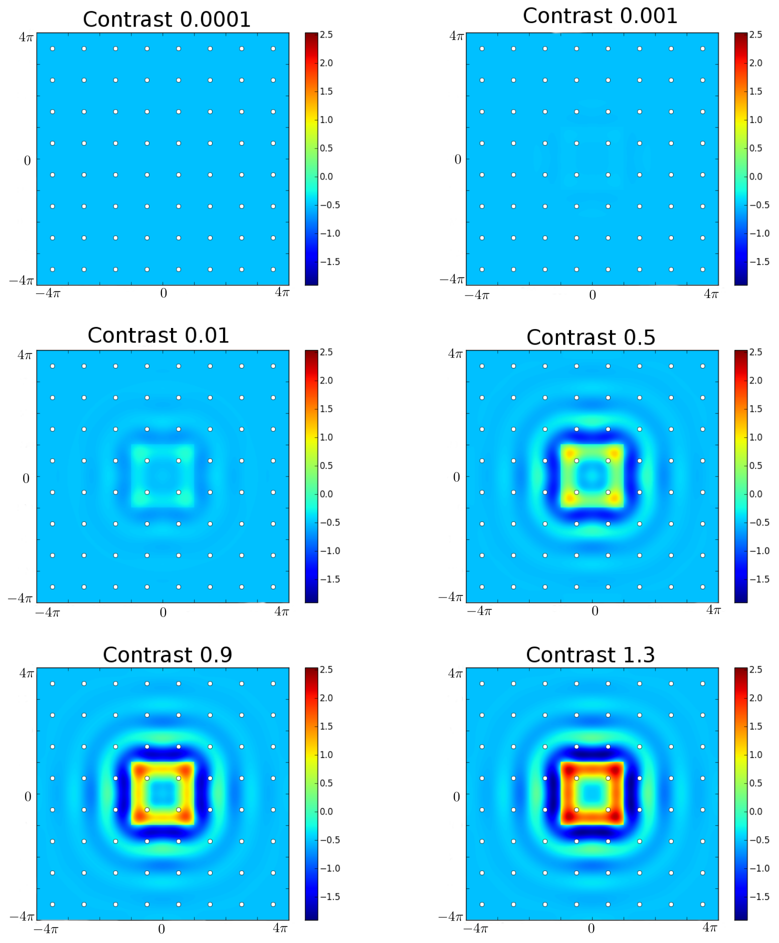


Figure 3.10: Final state of the network with contrast 1.0 and input orientation $\frac{\pi}{4}$. The white points indicate the positions of the pinwheel centers. the color code for all the plots is the same.

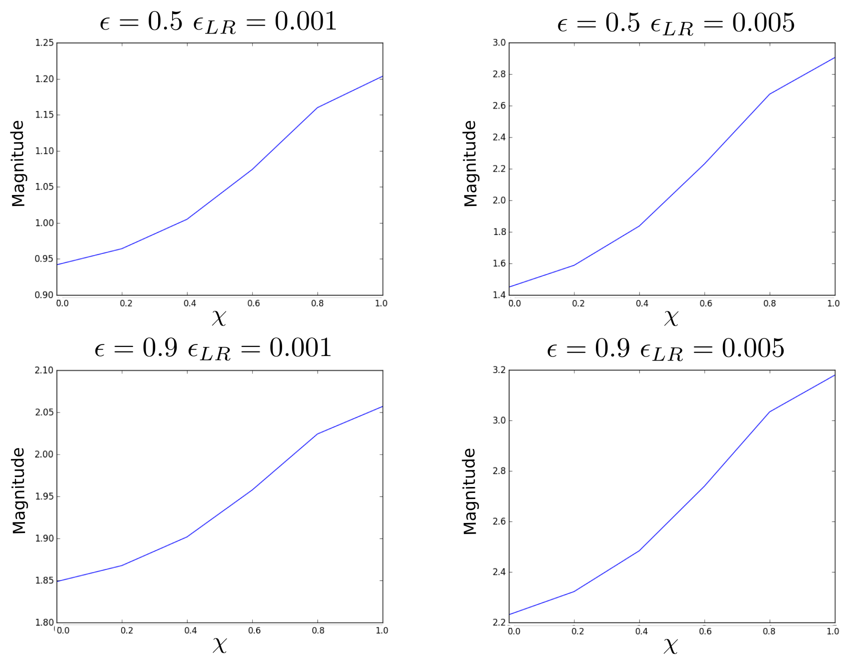


Figure 3.11: Modulus of the selection on different experiments with lateral connectivity. The results are shown as a function of the anisotropy

A problem that was encountered with these simulations is that with high levels of long range connectivity (ε_{LR}) and anisotropy (χ) the final selected angle was slightly shifted from the correct one. This small difference disappears when ε_{LR} is reduced to around 0.0025 and below. For example, when an input of $\frac{\pi}{2}$ is presented to a network with $\varepsilon_{LR} = 0.01$ and $\chi = 1$ the output angle is 1.25 instead of 1.57. The same network, if lateral connectivity is removed, outputs the correct orientation. An analytical study of this effect is currently in progress.

Finally, we computed the spontaneous activity state with lateral connections in a similar way as for the first experiment with this model. The results presented in figure 3.5 shows spontaneous activity organized in an hexagonal pattern. The structure of this solution is different from square distribution of pinwheels in the orientation map. When an input with low contrast is presented to the system it may try to move towards this hexagonal pattern instead of to the square one with the correct angle selection. For this reason, it may affect the selection capabilities of the network, or increase the contrast threshold at which the correct orientation is visible. Although this effect was not visible in the numerical simulations (see figure 3.6) it may affect the behavior of the model under certain configurations. The simulations with lateral connectivity, as presented in figure 3.12, show that this kind of connection changes the structure of the spontaneous activity to a square pattern which agrees with the pinwheel structure.

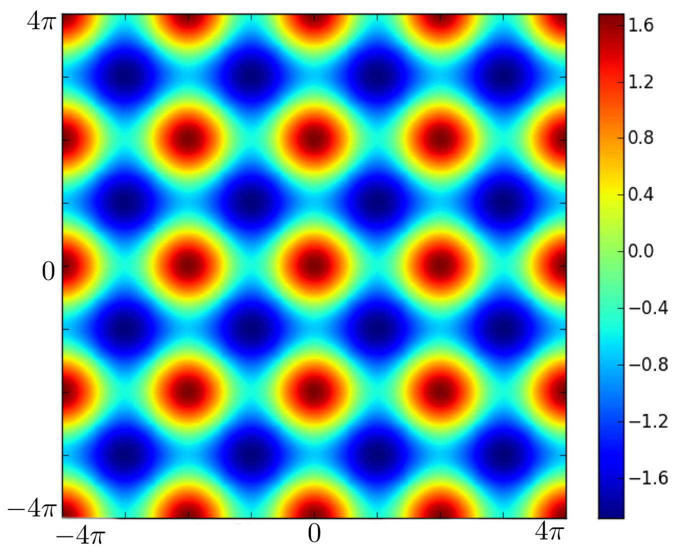


Figure 3.12: Spontaneous activity with lateral connection, $\chi = 0.5$ and $\varepsilon_{LR} = 0.01$

3.1.5 Discussion

In this section we have presented a package able to solve extremely fast the equations of a new neural field model of V1 for which no numerical simulations had been done before. It is, again, the computational power of GPUs that enables us to study the behavior of the system under a wide range of configurations. For these experiments the set of tools that are available together with the CUDA technology, were crucial, as two libraries were extensively used.

The simulations show that orientation can be represented in a neural field without explicit feature space. This approach is completely different from that of the well known Ring Model (see section 1.2.4.2) where the orientation preference is included in the model and the connectivity depends on this feature. This new approach is much more realistic and it can even be extended to use orientation maps obtained by biological experiments like the one in figure 1.5 whose pinwheel structure is more complex than a square grid.

The numerical simulations allow us to analyse two hypotheses proposed by Veltz in his original design of the model. First, he believed in the existence of two contrast thresholds that determine the behavior of the system. He predicted that if the contrast was below the first threshold the system would not do any selection, if it was between the first and the second it would select an angle different from the input orientations and if it was above the last value it would choose correctly. The experiments indicate that only the first limit exists.

A second prediction that was rejected by the numerical experiments was that localized input may produce a non local solution at low contrasts. This did not happen in the simulations as the expansion of the activity was always limited, agreeing with current biological experiments. This indicates the power of this modeling approach.

Future work concerning this model is to further analyze the effect of lateral connectivity on the angle selection capabilities. As it has been proposed in previous works [Bressloff 2001b], hallucinations or illusions may be produced by an increase in the strength of this connectivity. This kind of phenomena should also be produced in this approach. Also, it is possible to use this model with an orientation map obtained directly by optical imaging experiments. The data obtained in this kind of biological experiment is an orientation map with a structure more complex than the square grid used in this thesis. Simulations that incorporate real data can give us more information on the effect of the topology of the map in the orientation selection capabilities of the network.

3.2 A spatial extension of the Ring Model

3.2.1 Motivation

Another option to model the primary visual cortex as a neural field is to use the Ring Model of Orientation as a representation of a hypercolumn and then include as many of them as the amount of receptive fields in the visual area. This concept was originally used by Bresslof [Bressloff 2001b] and it is described in section 1.2.4.2. The analytical study of the model by the authors have showed that under certain conditions spontaneous patterns may appear that can be understood as visual hallucinations.

The analytical analysis presented in [Bressloff 2001b] and [Bressloff 2002a] shows that a sudden rise in the strength of the connectivity may produce patterns that correspond to known types of hallucinations. The authors mainly study the effect of long range connections in extreme cases where due to the existence of bifurcations the solutions of the system may change drastically. They do not provide simulations to show how the model behaves in a parameter range where the visual system works normally.

We believe that this model provides an excellent framework to study the effect of long range connectivity on the image processing capabilities of the primary visual cortex. These connections allow neurons to integrate information from areas far away from their receptive field, and, as has been suggested in [Field 2004], may be the cause of the contour integration capabilities of our visual system.

Gestalt psychologists proposed in the first half of the twentieth century a set of perceptual grouping principles that included the law of closure. This principle indicates that humans mind tends to see complete objects even if there are missing edges. For example, if some of the borders of a figure present gaps people are still able to determine its shape as if it was completely enclosed. One of the most classical example of this is the Kanizsa triangle illusion that can be seen in figure 3.13. The visual system is able to join the edges of the triangles and detect the complete figure.

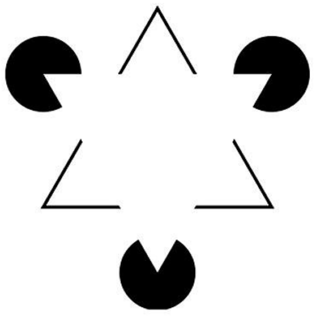


Figure 3.13: The Kanizsa triangle illusion shows how our mind is able to fill the gaps in a figure to detect a shape

In order to fill gaps, as in the Kanizsa triangle, the neurons with a receptive field corresponding to the missing parts of the shape need to integrate information from their neighbors. This can only be achieved by the effect of lateral connectivity between hypercolums as those included in this neural field model of the primary visual cortex. For this reason, we have ran several simulations, under different conditions, that have allowed us to determine if under this framework it is possible for lateral connections to induce this kind of perception.

3.2.2 Implementation

We have created the necessary software for doing the simulations that can show the effects of lateral connectivity. For this a system was built that first applied a set of oriented Gabor filters to a greyscale image and then uses the output of this operations as the input function for the neural field model. The integro differential equation is discretized and transformed into a set of ordinary differential equations that are solved in parallel in the GPU cluster. The procedure performed by the cards is similar to the one described in 2.2.1 for the Fokker-Planck equation. A diagram of this method is presented in figure 3.14

The selection of Gabor filters for an initial estimation of the position and orientation of edges is not random and is based on the hypothesis that simple cells receptive fields belong to this class of filter (see [Jones 1987]). Gabor filters are composed of a Gaussian kernel, known as the envelope, multiplied by a complex sinusoidal function, known as the carrier. We only consider the real component of the filter given by:

$$g(x,y) = e^{-\frac{x'^2+\gamma^2 y'^2}{2\sigma^2}} \cos(2\pi \frac{x'}{\lambda} + \psi), \tag{3.6}$$

where:

$$x' = x\cos\theta + y\sin\theta,$$

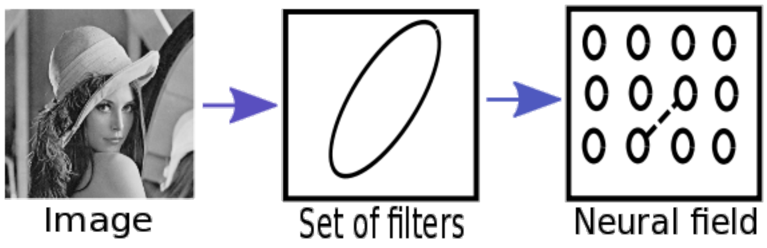


Figure 3.14: A diagram that represent the procedure done for the simulations. First a grey scale image is taken as input. Then a set of oriented Gabor filters are applied. The output of this step is then used as input for the neural field equations which are solved on the GPUs

$$y' = -x \sin\theta + y \cos\theta,$$

and λ, σ are parameters that control the shape of the filter. An example of a filter is presented in figure 3.15.

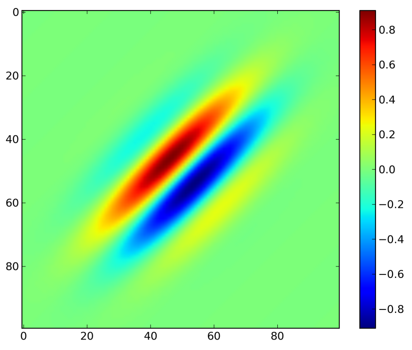


Figure 3.15: Example of a Gabor filter

One filter is applied to the image for each discretized orientation in the neural field model. This will give one resulting value per position and orientation that can then be used as an input function. Depending on the discretization this can require an amount of operations even bigger than solving the model equations. In all the simulations we have performed the size of the image was small enough to make the solution of the integro differential equation the bottleneck of the simulation and not the application of the filters.

To solve the neural field equations first the domain is discretized by setting the number of points in the space dimension equal to the number of pixels in the input image. This allow us to use the results from the filtering operations directly as an input. For the orientation variable a different number of points can be chosen but we used the same number as for the position. The equations are then divided equally between the GPUs available in the cluster and one thread is created for each point.

The discretization reduces the integro-differential equation to a system of ordinary differential equations (ODEs) that we approximate using the fourth-order Runge-Kutta method. This is the same procedure as for the discretized equations coming from the Fokker-Planck equation described in the previous chapter. The most computationally expensive step in solving this system is computing the right-hand side of equation (1.21), because two integrals, one of them being 3D, must be evaluated four times due to the integration scheme.

The integral corresponding to the lateral weight is the most complex element to compute in the right-hand side of the equations. In fact, if only local weights were used an efficient Fast Fourier Transform algorithm could be use to compute the integral term, as in the previous section. The complicated dependency of the long range connectivity to the orientation preferences prevents us from doing so.

The lateral weight function is normally assumed to be a Gaussian or a Mexican Hat function. We created a window for solving the second integral, whose size depends on the distance at which the weight function reaches 0. If no window existed, for each point it would be necessary to solve the integral over the whole domain, sampling from all points. Instead, with this technique, each thread samples only from a subset of points that are close to it.

The lateral weights are pre-computed at the beginning of the simulation and copied to the GPU memory. This is a 4 dimensional matrix of size (number of orientations)^2 $\times$ (number of points in the window)^2 as it must consider the difference in preferred orientation, the distance, and it should expand in the corresponding angle. For computing the integral each thread must iterate over a 3 dimension subset of this matrix which depends on its orientation.

To make this access faster we copy this matrix into the shared memory by parts as it is too big to fit in this small amount of space. The small amount shared memory on each card is extremely fast and can be accessed by all the threads in the same block. First we make sure that all the threads in each block have the same orientation preference, so they can share the same 3-dimensional subset of the original matrix. This is simple to do as the angle is considered the first dimension of the array containing the points and any block size which is a divisor of the total number of pixels will work.

The iterations for computing the integral must sample all the neighbors and all the preferred orientations at those positions. For this, the threads in each block will first copy a part of the weight matrix that corresponds to the values for the common orientation preference and one different angle (equal for all the threads). This is a small 2 dimensional matrix of size (number of points in the window)^2 which fits in the reduced space. This procedure is done in parallel, as each thread copies a different part of the matrix from global memory to shared memory. Once the copy is finished the threads synchronize, using the GPU primitives for block organization, and then sample all of their neighbors at the corresponding external angle and multiply the values by the weight matrix. This procedure is repeated for each possible angle. A pseudo-code for the process is presented next. In the code w_shared is a linear array in shared memory, lateral_weights is a matrix in global memory that contains the pre-computed lateral weights and sample is a function that computes the multiplication of the weights with the results of the sigmoid. Also, local_angle is the common orientation preference for all the members of the block and thread position correspond to the coordiantes of the point for which the current thread is computing the righ-hand side.

Algorithm 2 Computation of the integral with shared memory

```
for each possible angle $\phi$ do
    w_shared[thread_id] $\leftarrow$ lateral_weights[local_angle,$\phi$,thread position]
    synchronize the block
    integral = integral + sample(w_shared,S(V))
    synchronize the block
```

This procedure adds some instructions and synchronization to the code but makes the access to a lot of data that is common to all the threads faster. Each thread in a block will need to access all the elements of this reduced weight matrix once so the total amount of global memory access is greatly reduced. Access to shared memory is several orders of magnitude faster than access to global memory.

Another way of improving the memory access of the simulations is to choose a block structure which will allow coalesced memory access. When a GPU does a memory access it will read not one value, as in a standard processor, but several continuous elements. If neighbor threads need neighbors elements of memory the values for several of them can be obtained with just one memory access. If this doesn't happen some threads will need to wait until the addresses of memory required can be fetched and the extra values read each time will be lost. In our simulations if the block size is chosen to be exactly the number of points on one row this kind of memory access will be enhanced.

If all the threads in a block are members of the same row they will all be adjacent in the array and because of this they will always access consecutive elements in the input data. This is represented in the diagram of figure 3.16 where each square represents a lateral weight window for different points. It is easy to notice that they are moved by exactly one position so that they will always need neighboring elements. This configuration also enables cache memory to work efficiently as, for example, the first element to be read by the green thread is the second element to be read by the red thread and this value will be kept in cache after the first access.

An even faster simulation can be obtained if the number of points in one row of the image is a multiple of the warp size of the card, normally 32. This will ensure that all the processors in the GPU are always used. If this doesn't happen one warp will be incomplete and processors will wait for the others to finish.

We have measured the execution time for a 104×104×104 grid with a lateral window size of 20×20. Figure 3.17 shows the mean execution time for one time step measured after 100 executions. The plot shows how as more cards are added

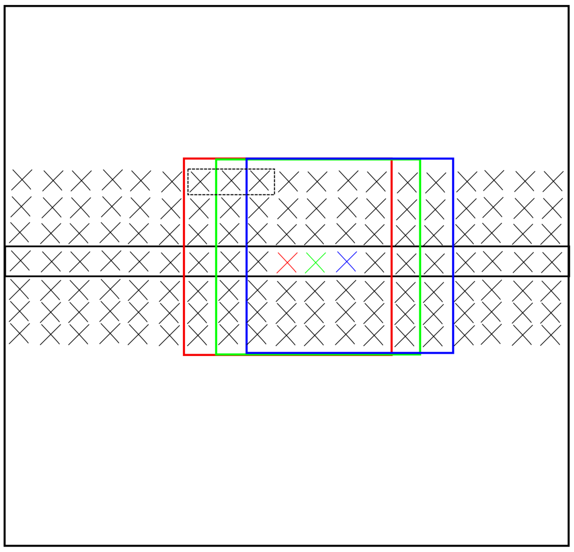


Figure 3.16: A diagram that represents coalesced memory access when the block size is equal to the number of points in one row. Each thread/point is represented by an X and each colored rectangle represents the window for the lateral weights of one consecutive point in the block (also marked with the same color). The small rectangle with dashed line indicates values that can be fetched in just one memory access.

the faster the simulation becomes. With 7 cards (maximum in one computer) the simulation reaches 0.88 seconds per time step. The system was not tested with more cards to avoid MPI communication.

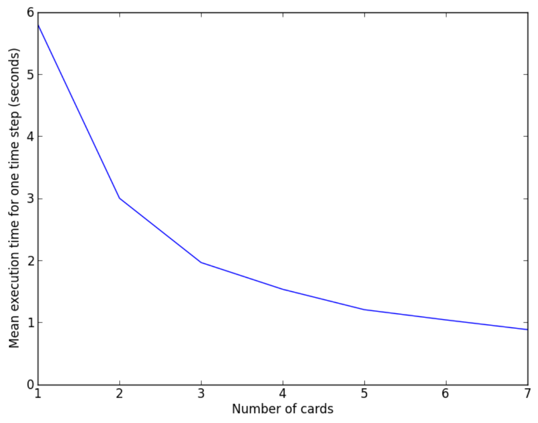


Figure 3.17: Time for the execution of one time step with different number of cards

To compare this result a second implementation was created in which the GPU kernel was replaced by a for loop. This procedure is the same as the one done for the Fokker-Planck solver presented in section 2.2.4. The parallel computation normally done in the card is now directly performed in each processor. A comparison with this version allows us to approximate the size of a standard high performance computing solution that would be needed to reach the same speed up. One time step of the simulation using this second code and 10 processor took 143.56 seconds which is 163 times slower than with 7 GPUs. This experiment was done in the same computer, but more processors were used in the GPU simulations.

A previous version of these results, with am earlier, slower implementation, was reported in [Baladron 2012a].

3.2.3 Results

We have performed several experiments to test the effect of lateral connectivity. One difficulty for doing this is that the anisotropy of the long range connections generates a problem with the discretization technique. Due to the square shape of the image some orientations will connect with more elements than others. For example, if a line is drawn through the diagonal of the square it will pass through more points than any other. A diagram showing the expansion of two orientations starting from the same pixel is shown in figure 3.18, one line connects with two other position while the others connects with 0. This problem can't be solved by increasing the number of points as it is a property of the shape of the domain.

If this problem is not solved the system will have a bias that will make the hypercolumns almost always choose an angle close to the orientation with more points.

Even if the connectivity values in that direction are smaller than in the correct one they will sum over more elements obtaining a larger value. To compensate for this we have normalized the lateral connectivity depending on the number of elements the point connects to. This can be pre-computed and used during the simulations.

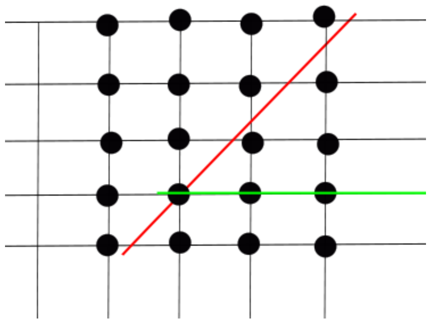


Figure 3.18: Diagram showing the problem of the anisotropy of the connectivity with the discretized domain. The horizontal orientation connects with more points than the oblique one.

We have tested the system with two binary images, one is similar to the Kanisza triangle presented in figure 3.13 and the other is a square with gaps. The objective of the experiments was to determine if long range connectivity was enough to activate the elements whose receptive field correspond to the gaps in the figure. The continuation given by the connectivity should activate those zones and provide higher cortical areas with an input that indicates the existence of the triangle or square. The results for the square are presented in figure 3.19 and for the triangle in figure 3.20. The image on the top is the input and the two one on the bottom are the final state of the network, after convergence, and a thresholded version of the stationary solution. Clearly, as expected, there is activity in the zones where no input is received.

We also created a different weight function from the one in (1.23), based on the cocircularity principle. Two edges are cocircular, as defined in [Parent 1989], if there exist a circle for which both of them are tangent. In [Geisler 2001, Sanguinetti 2010] it is shown that the statistics of edge appearance on natural images can be explained by cocircularity. The brain is trained with this kind of input and probably learns its connectivity with respect to the real probability of finding edges at different position and orientations. Figure 3.21 shows the weight function for an element at the center of the image with preferred orientation 0. The value shown in the figure is for the orientation with the maximum weight at each position. The shape of the function is given by cocircularity and the size by the distance. The results with this connectivity are presented in figure 3.22. The extension of the edges is still produced but activity at the same level as in the gaps also appears around the original lines, making the continuation effect less clear. This is produced because the new connectivity connects one pixel with a larger amount of elements than the

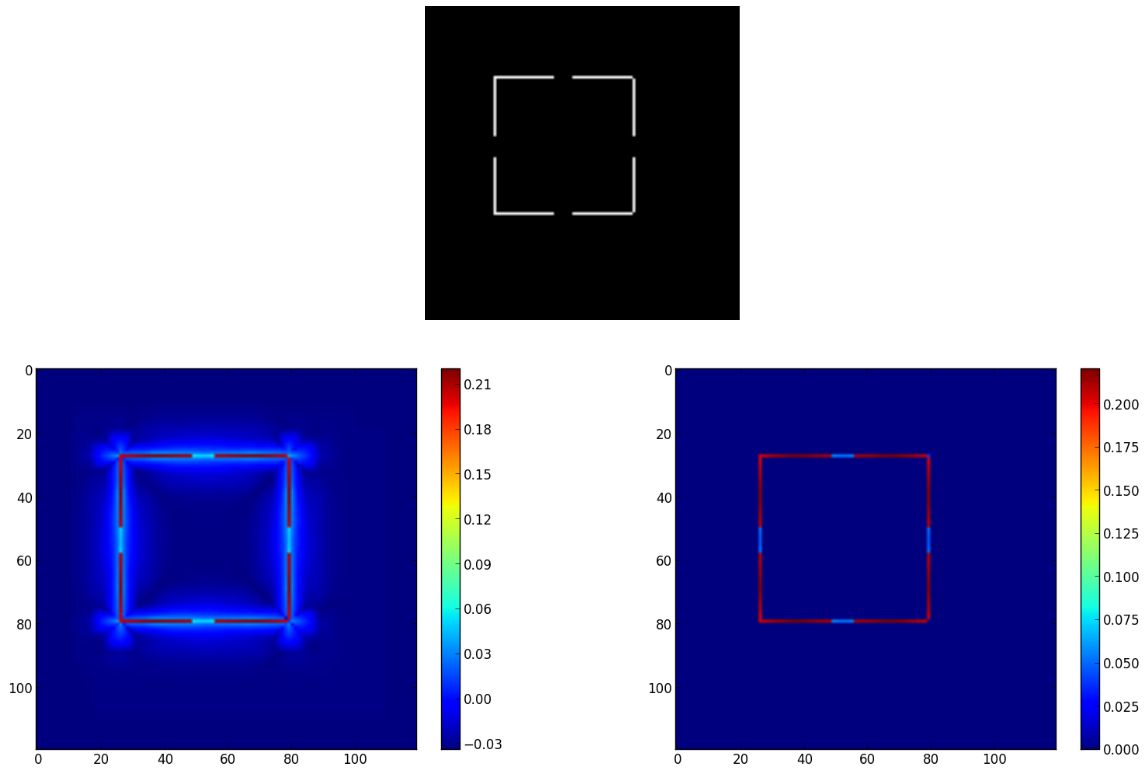


Figure 3.19: Top: second input image to the system. Bottom: stationary solution and thresholded version of the output

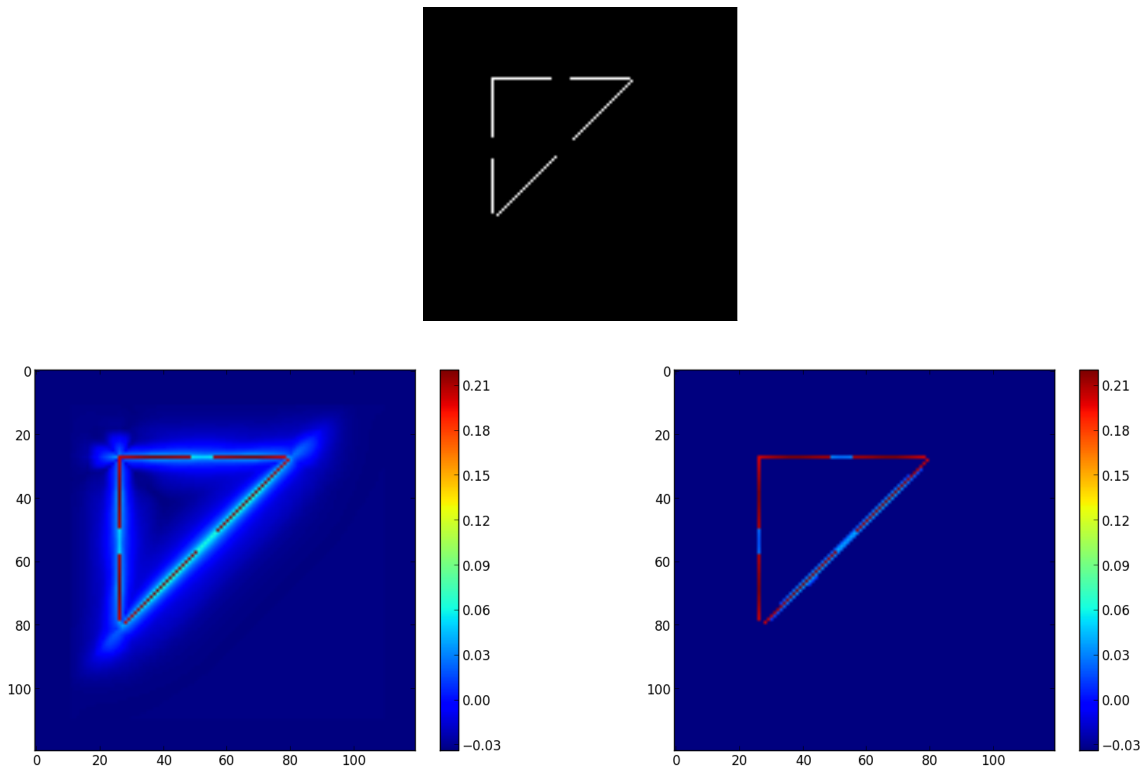


Figure 3.20: Top: first input image to the system. Bottom: stationary solution and thresholded version of the output

previous weight function.

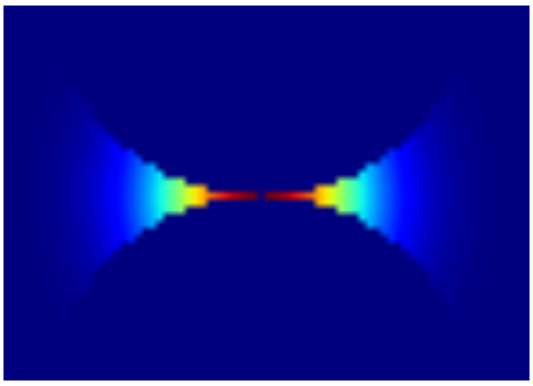


Figure 3.21: Maximum weight for each position for an edge with orientation 0 at the center of the image.

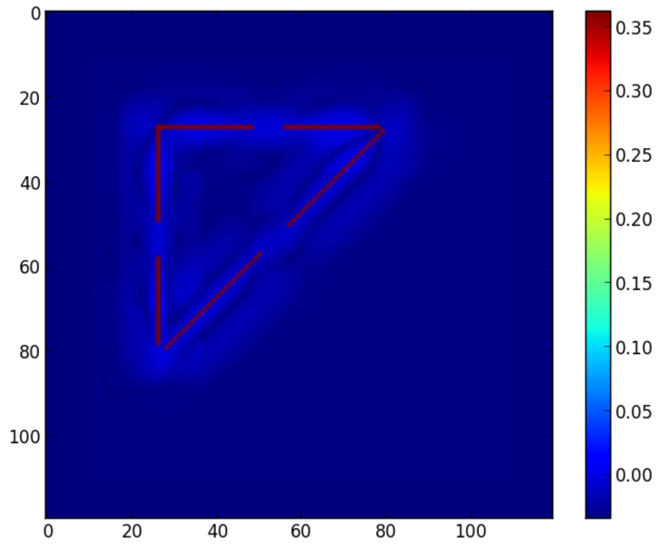


Figure 3.22: Result for a simulation with the triangle input from figure 3.20 and cocircularity weights.

We have also tested the system with some real images and with both weight functions, without great success. In the majority of the cases the expansion of the edges spreads over the whole image, activating almost every point. The amount of edges and different orientations in real images is much wider that in our synthetic example. One possible solution to this is the inclusion of inhibition, this should reduce the overall activity, stopping the expansion. We also included this feature, giving the lateral weights a difference of Gaussian shape (Mexican hat). Our experiments indicated that neither short range enhancement - long range inhibition nor short range inhibition - long range enhancement performed a correct edge continuation. A more extended search in the parameter space of the model with the help of analytical techniques like the ones presented in [Veltz 2011] could possibly lead to improved results.

3.2.4 Discussion

In this section a system designed to test the capabilities of long range connections in the primary visual cortex was presented. The software developed is extremely fast and its speedup is similar to a standard high performance computer solution with hundreds of processors. The good use of memory in the GPU cluster in which the simulations were developed provided an extremely fast code that can be extensively used by modelers studying the mesoscopic behavior of the primary visual cortex and other areas.

An initial set of experiments shows that the law of good continuation and closure described by Gestalt psychologists can be implemented by the shape of the long range connectivity in the primary visual cortex. The simulations indicate that neurons with a receptive field corresponding to a illusory contours, like the ones of the Kanizsa triangle, can be activated by lateral connectivity, providing a perception of the complete figure. These results are not perfect as the expansion of the edges is on several directions and not only in the ones with a gap. This can probably be controlled by providing correct inhibition. Our experiments with the model, shows that obtaining the correct inhibition-excitation configuration is a hard task and probably requires a more rigorous analytical treatment.

The results with real images are disappointing as no good enhancement of edges could be obtained. The expansion of the information in all directions made the network explode with activity for all the configurations we tested. Inhibition could stop the expansion but it also destroyed the contour integration capabilities. We believe than finding the correct parameter range is a very complex task that needs to be studied with the help of other tools.

Probably the most significant result of this section is the software itself. Analytical treatment of neural field equations is normally not associated with a numerical evaluation of the predictions made by theory. This is normally due to the computational complexity of the simulations. Phenomena such as spontaneous patterns or short term memory (bumps) can now be produced numerically by expert neuroscientists. The implementation of Gestalt rules by long range connectivity is just an example of the possible biological phenomena that this modeling technique can account for and allow to explore further.

Chapter 4
General conclusion

In this Thesis we have used mesoscopic models and high performance computing to study the dynamics of large realistic neural networks. We have focused our analysis on models that can be used to explain how edge selection is implemented in the primary visual cortex. In the first part of this Thesis we have used a mean-field reduction, that allows us to transform a large system of stochastic differential equations into a partial differential equation, to study the behavior of multipopulation networks. In the second part of this Thesis we have studied two neural field models that allow us to look at the cortical activity at a different scale.

In the part concerning the mean-field reduction we have created the necessary software for solving the McKean-Vlasov-Fokker-Planck equation, that describes the probability density function of a network of FitzHugh-Nagumo or Morris-Lecar neurons, on a GPU cluster. The speedup obtained with our implementation is equivalent to what can be obtained with a cluster with hundreds of processor (assuming linear behavior). We have used this code to study the effects of noise on large ensembles of neurons and to study a multipopulation representation of a hypercolumn from the primary visual cortex.

Our numerical experiments have shown that for a large population of neurons larger levels of noise may increase the convergence speed of the probability density towards a stationary distribution. The solution is similar for all noise levels, indicating that this effect may be used for conveying information. In fact, we believe that a sample based representation of the stationary distribution can be created in a postsynaptic population if the connectivity is dense enough. If this is the case, the time required for the transfer of information would be equal to the one required for convergence, that can be tuned by external noise.

Another group of numerical experiments showed that the convergence speed may also be tuned by increasing the value of the input function. This is a similar effect to the one produced by the external noise level but with another parameter of the model. In this case the solutions are different with higher input values producing larger firing rates. As one of our assumptions is big levels of noise, it is very difficult for low levels of input to exist as this value represents the sum over the activity of previous areas which also present noise.

We have also extended the previous simulation to a multi-population network. For this we used a model of one hypercolumn of V1 where each possible orientation is represented by two populations, one excitatory and one inhibitory. The connectivity in the model depends on the angle difference. The previous relation between external noise and convergence speed still existed in this experiments, but when the noise is low oscillations may appear.

The model is able to enhance an initial, weakly tuned, orientation selection represented by its input function. The output of the populations show a clear angle selection that can be used by higher processing areas. In order to reach this objective the inhibition level is extremely important as it regulates the decrease in the activity of orientations far from the input angle. This shows how the brain may implement computations using a population in the mean field limit as a basic unit instead of a simple neuron.

In the second part, considering the neural field models of the primary visual cortex, we first showed a numerical analysis of a new representation of V1 where the orientation preference is not a feature of the equation. In this approach an orientation map is used to create an input function that represent an initial edge selection. Our numerical experiments showed that with a square grid of pinwheels and only local connectivity (distance dependent) the model is able to sharpen the initial edge selection. It also showed the existence of a contrast threshold at which the selection is correct. The results are different from the predictions made in the original proposal of the model [Veltz 2011] where the existence of two different thresholds was considered. This is the first time that the capacities of this system are evaluated numerically.

Another prediction made in the original proposal of the model was that localized input could produce a non-local output. The numerical experiments showed that independent of the contrast, if input was only present on the central pinwheels, the output had a limited expansion. This agrees with biological data [Chavane 2011] and provides a starting point for further analysis.

We have also tested the effect of patchy long range lateral connections on the model. The experiments showed that the response is improved when the anisotropy is increased. This agrees with both the original predictions and biological data that indicate that lateral connections follow the direction of the orientation preference.

In a second group of experiments, with a different neural field model of the primary visual cortex, we have showed how long range connections could be used for processing illusory contours. For this, we have run a set of simulations that use as input a figure similar to the Kanisza triangle and output a matrix of cortical activity where the neurons with a receptive field corresponding to a missing edge are active, indicating that a line should be present at that point.

All these simulations were made on a GPU cluster and a set of techniques for a fast implementation were proposed. The neural field equations are easy to solve if the weight function depends only on the distance between points, because in this case the integral term correspond to a convolution that can be solved using a Fast Fourier Transform. With a more complex connectivity and realistic this is impossible and a good usage of the shared memory of the GPUs is required. We proposed a way to group points with similar weights in each block, so that, at the beginning of each iteration for the computation of the integral, a common matrix can be copied to the shared memory and provide a fast access. As in the previous case, the speedup obtained is equivalen to that obtained by a standard cluster with hundreds of processors.

Perspectives

The numerical methods for solving the McKean-Vlasov-Fokker-Planck equation can be improved by introducing a multi-grid approach to the relaxation technique that was proposed in this Thesis. This can be used together with an adaptive mesh algorithm that can create a grid with a bigger amount of points close to the limit cycle and less outside. This kind of methods are complex to implement and parallelize but there are projects where software packages have been created implementing these techniques and used for different types of partial differential equations. We believe that the next step in order to achieve an even faster solver for the Fokker-Planck equation is to use these libraries on large clusters.

An important step further in this research would be to improve the mean field reduction in order to include more complex weight matrices. The current approach assumes a common weight between all neurons in the same population which may not be realistic in many cases. Also, learning or changes in the weight are not consider and can be extremely important for certain computations in the brain. Once a new set of equations is obtained the reduction can be used as a simulation tool in a similar manner as it has been done in the Thesis.

We believe that a better understanding of the consequences on the information processing capabilities of the cortex of the propagation of chaos effect together with the existence of a stationary probability density is required. This feature of the kind of network studied in the first part of the thesis may be used for fast computations if the neurons may implement methods like maximum likelihood to estimate the probability density of the presynaptic population using the samples obtained through synapses. It is not clear how this can be done under biological constraints, but this can open a completely new perspective on information coding.

Regarding the second part of this Thesis, an analysis of the bifurcations of the neural field model presented in section 3.1 is required to better understand the conditions at which a correct angle selection is obtained. The simulations presented in this work have increased the knowledge on the dynamics of the equations and have shown that a better analytical study is required. Also, the experiments have shown the existence of hexagonal patterns of spontaneous activity that may influence the final solution of the system. This require complex mathematical techniques and is a work currently being done by other researchers.

The spatial extension of the Ring Model presented in 3.2 can be improved in order to process real images instead of the simple Kanizsa triangle used in this Thesis. The edge detection capabilities of the network may be exploited by any image processing system. For this, the inclusion of a correct level of inhibition is crucial as wrong edges must be deleted and also the extension of lines must be limited. Analytical techniques may be required in order to find the correct parameter range for the system to process more complex images..

The mean field reduction together with a series of simulations is published in [Baladron 2012b]. An initial description of the multi-GPU implementations of the models is published in [Baladron 2012a].

Chapter 5

Conclusion générale (version française)

Dans cette thèse nous avons utilisé des modèles mésoscopiques et de calcul de haute performance pour étudier la dynamique de grands réseaux neuronaux réalistes. Nous avons centré notre analyse sur des modèles qui peuvent être utilisés pour expliquer comment les contours peuvent être sélectionnés dans le premier cortex visuel. Dans la première partie de cette thèse nous avons utilisé une réduction de champs moyens qui nous permettent de transformer un grand système d'équations différentielles stochastiques en une équation différentielle partielle pour étudier le comportement de réseaux avec des populations multiples. Dans la seconde partie de cette thèse nous avons étudié deux modèles de champs neuronaux qui nous permettent d'observer l'activité corticale à différentes échelles.

Dans la partie concernant la réduction de champs moyen, nous avons créé le logiciel nécessaire pour résoudre l'équation de McKean-Vlasov-Fokker-Planck , qui décrit la fonction de densité de probabilité d'un réseau de neurones de FitzHugh-Nagumo ou Morris-Lecar dans un grappe de GPU. La vitesse obtenue avec notre implémentation est équivalente à celle qui pourrait être obtenue avec un grappe de centaines de processeurs (en assument un comportement linéaire). Nous avons utilisé ce code pour étudier les effets du bruit dans de grands groupes de neurones et une représentation d'une hypercolonne du premier cortex visuel.

Nos expériences numériques nous ont montré que pour une grande population de neurones de grands niveaux de bruits peuvent augmenter la vitesse de convergence de la densité de probabilité vers une distribution stationnaire. La solution est similaire pour tous les niveaux de bruits indiquant que cet effet peut être utilisé pour transférer des informations. D'ailleurs nous croyons qu'une représentation basée sur des échantillons peut être créée par une population postsynaptique si la connectivité est suffisamment dense. Dans ce cas le temps nécessaire pour transférer les informations serait pareil à celui nécessaire pour la convergence, qui peut être ajustée par le bruit externe.

Un autre groupe d'expérience montre que la vitesse de convergence peut être aussi améliorée en augmentant la valeur d'entrée. Ceci est un effet similaire à celui produit par des bruits externes mais avec d'autres paramètres du modèle. Dans ce cas les solutions sont différentes avec des valeurs d'entrées plus grandes qui produisent de plus grands taux d'activation. Comme le modèle assume de grands niveaux de bruits, il est très difficile qu'il existe de bas niveaux d'entrée.

Nous avons étendu nos simulations à un réseau avec de multiples populations.

Pour cela nous avons utilisé un modèle d'une hyper colonne du premier cortex visuel où chaque orientation possible est représentée par deux populations une excitatrice et une inhibitrice. La connectivité du modèle dépend de la différence d'angle. La relation antérieure entre le bruit externe et la vitesse de convergence existe toujours mais quand le bruit est bas il se peut qu'apparaissent des oscillations.

Le modèle est capable d'améliorer une sélection d'orientation initiale. Le output de la population montre une sélection d'angle plus claire qui peut être utilisée par des aires de traitement de plus haut niveau. Pour réussir cet objectif le niveau d'inhibition est extrêmement important puisqu'elle régule la diminution de l'activité des orientations éloignées à l'angle d'entrée. Cela montre comment le cerveau peut mettre en place des calculs en utilisant des populations dans la limite des champs moyens comme une unité basique au lieu d'un seul neurone.

Dans la deuxième partie qui considère les modèles de champs neuronaux, d'abord nous avons montré une analyse numérique d'une nouvelle représentation du premier cortex visuel d'où la préférence d'orientation n'est pas une caractéristique de l'équation. Dans cette approche une carte d'orientation est utilisée pour créer une fonction d'entrer qui représente une sélection initiale de contours. Nos expériences numériques ont montré qu'avec une grille carrée de pinwheel et seulement avec des connectivités locales (selon la distance) ce modèle est capable d'améliorer une sélection initiale de contours. Aussi elles ont montré l'existence d'une limite de contraste à partir duquel la sélection est correcte. Ce résultat est différent aux prédictions faites dans la proposition initiale du modèle ([Veltz 2011]) où l'on croyait qu'il y avait deux limites différentes. C'est la première fois que les capacités de ce système sont évaluées numériquement.

Une autre prédiction faite dans la proposition initiale du modèle est qu'une entrée localisée peut provoquer un output non localisé. Les expériences numériques ont montré qu'indépendamment du contraste si le input se présente seul devant les pinwheel centrales l'output a une expansion limité. Cela concorde avec des données biologiques ([Chavane 2011]) et donne un point de départ pour les analyses futures.

Nous avons également essayé l'effet des connexions latérales dans le modèle. Les expériences ont montré que la réponse s'améliore quand l'anisotropie est augmentée. Cela concorde avec les prédictions initiales et les données biologiques qu'indiquent que les connexions suivent la direction de l'orientation préférée par le neurone.

Dans le deuxième groupe d'expérience avec un autre modèle de champs neuronaux nous avons montré que les connexions de longue distance peuvent être utilisées pour le traitement de contours illusoires. Pour cela nous avons un ensemble de simulations où nous avons utilisé une figure similaire au triangle de Kanizsa et nous avons obtenu de l'activité dans les neurones correspondant à un contour manquant. Cela indique qu'une ligne doit être présente sur ce point.

Toutes ces simulations ont été réalisé dans un cluster de GPU. Un ensemble de techniques ont été proposées pour une mise en place rapide des modèles. Les équations des champs neuronaux sont faciles à résoudre si la fonction de poids dépend uniquement de la distance entre chaque point. Dans ce cas l'intégrale correspond à une convolution qui peut être résolue avec une transformé rapide de Fourier.

Avec une connectivité plus complexe et réaliste cela est impossible. On a besoin d'une bonne utilisation de la mémoire partagée de la GPU. Nous avons proposé une manière d'agrouper des points avec des poids similaires dans chaque bloc. Au début de chaque itération pour le calcul de l'intégrale une matrice commune est copiée à la mémoire partagée pour permettre un accès rapide. Comme dans le cas précèdent la vitesse obtenue est équivalente à un cluster avec des centaines de processeurs.

Perspectives

Les méthodes numériques pour résoudre l'équation de McKean-Vlasov-Fokker-Planck peuvent être améliorés en utilisant un algorithme de multigrid. Cela peut être utilisé avec un algorithme de grille adaptative qui soit capable d'utiliser plus de points prêt du cycle limite et moins dehors. Ce type de méthode est difficile à mettre en place et à paralléliser mais il existe des projets où l'on a créé des logiciels capables d'utiliser ces techniques pour différents types d'équations différentielles partielles. Nous croyons que le prochain pas pour obtenir un solutionneur plus rapide est d'utiliser cette bibliothèque dans de grands clusters.

Un pas important dans cette recherche serait d'améliorer la réduction de champ moyen pour inclure des matrices de poids plus complexes. La méthode actuelle assume un poids identique entre tous les neurones de la même population. Cela n'est pas réaliste dans beaucoup cas. Aussi l'apprentissage ou changement de poids ne sont pas considérés. Une fois qu'un nouvel ensemble d'équation est obtenue la réduction peut être utilisée comme un outil de simulation tel comme elle a été faite dans cette thèse.

Nous pensons qu'une meilleure compréhension des conséquences dans les capacités de traitement des informations du cortex de l'effet de propagation du chaos et l'existence d'une densité de probabilité stationnaire est réquisitionnée. Cette caractéristique du type de réseau étudié dans la première partie de cette thèse peut être utilisée pour computer rapidement si les neurones peuvent mettre en place des méthodes tel que maximum likelyhood pour estimer la densité de probabilité de la population présynaptique en utilisant des échantillons obtenues par le biais des synapses. On ne sait toujours pas comment cela peut être réalisé sous des restrictions biologiques. Cela peut ouvrir une perspective totalement neuve sur la codification d'information.

Dans la deuxième partie de cette thèse une analyse des bifurcations du modèle de champs neuronaux se requiert pour mieux comprendre les conditions dans laquelle une sélection d'angle correct peut être obtenue. Les simulations présentées dans ce travail augmentent notre connaissance de la dynamique de l'équation et ont montré qu'une meilleure étude analytique est nécessaire.

Aussi les expériences ont montré l'existence d'un patron hexagonal d'activité spontanée qui peut influencée la solution finale du système. Cela nécessite des techniques mathématiques complexes et c'est un travail qui est actuellement réalisé par d'autres chercheurs.

L'extension spatiale du ring model présenté dans 3.2 peut être améliorée pour traiter des images réelles au lieu du simple triangle de Kanise utilisé dans cette thèse. Les capacités de détections de contours de réseaux peuvent être exploitées par n'importe quel système de traitement d'image. Pour réussir cela l'inclusion d'un niveau correct d'inhibition est cruciale puisque les contours incorrects doivent être effacés et les extensions de lignes doivent être limitées. On a besoin de techniques analytiques pour trouver la gamme des valeurs pour pouvoir traiter des images plus complexes.

La réduction de champs moyens avec une série de simulation est publiée dans [Baladron 2012b]. Une description initiale de l'implantation multi GPU des modèles est publié dans [Baladron 2012a].

Appendix A

Numerical methods for differential equations

Contents

A.1 Numerical methods for ordinary differential equations

In the first part of this appendix we introduce the methods for solving numerically Ordinary Differential Equations (ODEs) that are used in this thesis. The motivations for using ODE solvers are twofold. First, in chapter 2 a Partial Differential Equation (PDE) is solved using the method of lines. This transforms the equation into a set of ODEs that are solved with some of the methods presented in this Appendix. Second, in Chapter 3, the integrodifferential equations of some neural field models are solved by transforming them to ODEs. An understanding of the topics covered in this part of this appendix will allow the reader to comprehend how the equations used in this thesis are solved and also why in some of the experiments they become unstable.

For more details on the numerical methods and its analysis we recommend the books [Hairer 2008] and [Hairer 2010].

A.1.1 Initial value problem

An initial value problem is composed of a differential equation and an initial condition. The objective is to solve the equation:

$$\frac{dy(t)}{dt} = f(t, y), \tag{A.1}$$

given the initial point (t_0, y_0) and where $y : \mathbb{R} \to \mathbb{C}^n$ and $f : \mathbb{R} \times \mathbb{C}^n \to \mathbb{C}^n$

An analytical method for solving this kind of problem will provide an expresion for $y(t)$ that fulfills the contraint that $y(t_0) = y_0$. A numerical method will provide an approximation for the values of $y(t)$ at some values of t. Normally, this is enough information for most analyses when finding an explicit formula is too complicated or impossible.

A.1.2 Euler's method

Euler's method will provide us with the values of y at different values of t. The method start with $t = t_0$ and at each step computes an approximation of y at the current time plus Δt, a small value called the time step size. The method starts at the point (t_0, y_0), then moves to $(t_{1,}y_1) = (t_0 + \Delta t, y(t_0 + \Delta t))$, then to $(t_2, y_2) = (t_1 + \Delta t, y(t_1 + \Delta t))$, and so on, until a maximum time, T, has been reached.

One derivation of the methods uses the definition of the problem together with the formula for the slope of a line given two points to obtain the following relation:

$$\frac{y_{k+1} - y_k}{t_{k+1} - t_k} = f(t_k, y_k)$$

As $t_{k+1} - t_k = \Delta t$ we can express the previous equation as:

$$y_{k+1} = y_k + f(t_k, y_k)\Delta t. \tag{A.2}$$

Equation (A.2) provides a rule for the iterations of Euler method. To use it, we first compute the value of the function f and then we use this result to obtain an approximation of the next point.

Another possible derivation of the method uses the Taylor expansion of y around t_0, given by:

$$y(t_0 + \Delta t) = y(t_0) + \Delta t y'(t_0) + \frac{1}{2}\Delta t^2 y''(t_0) + O(\Delta t^3). \tag{A.3}$$

Replacing y' with $f(t, y)$ and ignoring the high order term, the same formula as in Equation (A.2) can be obtained.

The error of one step of the method, called the local truncation error, is equal to the sum of the all the high order terms of Equation (A.3), this value is proportional to Δt^2. The error after achieving time T, called the global truncation error, depends on the number of time steps used, which is equal to $\frac{T-t_0}{\Delta t}$, a value porportional to $\frac{1}{\Delta t}$. Finally, by multiplying the order of the local error $(O(\Delta t^2))$ with the previous value, we obtain a global error of $O(\Delta t)$.

A.1.3 Runge-Kutta methods

Although Euler's method is simple to use, its global truncation error is high. One way to improve the accuracy of the method is to use the slope of more points in the interval $[t_k, t_{k+1}]$, instead of only the two extreme ones. When this is done, the methods is said to belong to the Runge-Kutta family.

The most famous method from this familiy is Runge-Kutta 4 or RK4, which is defined by the following equation:

$$y_{k+1} = y_k + \frac{1}{6}\Delta t(k_1 + 2k_2 + 2k_3 + k_4), \tag{A.4}$$

with:

$$\begin{aligned}
k_1 &= f(t_k, y_k)\\
k_2 &= f(t_k + \frac{1}{2}\Delta t, y_k + \Delta t\frac{1}{2}k_1)\\
k_3 &= f(t_k + \frac{1}{2}\Delta t, y_k + \Delta t\frac{1}{2}k_2)\\
k_4 &= f(t_k + \Delta t, y_k + \Delta t k_3).
\end{aligned}$$

This method approximates the function at the next time step by computing a weighted average between four increments based on the slope at different points of the interval $[t_k, t_k + \Delta t]$.

The local truncation error is $O(\Delta t^5)$ and the global truncation error is $O(\Delta t^4)$.

A general s-stage fixed time step Runge-Kutta method may be written as:

$$y_{k+1} = y_k + \Delta t \sum_{i=1}^{s} b_i k_i, \tag{A.5}$$

$$k_i = f(t_k + c_i\Delta t, y_k + \Delta t \sum_{j=1}^{i-1} a_{ij} k_j), \tag{A.6}$$

where a_{ij} and b_i are parameters of the method and .

$$c_i = \sum_{j=1}^{i-1} a_{ij},$$

A.1.4 Stability analysis

Lets start with a very simple equation:

$$\frac{dy}{dt} = \lambda y, \tag{A.7}$$

with $\lambda \in \mathbb{C}$. The solution, $y(t) = y_0 e^{\lambda t}$, to this equation converges to 0 when time goes to infinity, whatever the initial condition, if $Real(\lambda) \leq 0$.

If we use Euler's method to solve it numerically, we will get the following iterative rule:

$$y_{k+1} = y_k + \Delta t \lambda y_k,$$

which can also be expresed as:

$$y_{k+1} = (1 + \Delta t \lambda) y_k = R(\Delta t \lambda) y_k,$$

where $R(z)$ is the polynomial $1+z$. This recursive equation can be solved to obtain:

$$y_{k+1} = R(\Delta t \lambda)^{k+1} y_0. \tag{A.8}$$

The value y_{k+1} will remain bounded when $(k+1) \to \infty$ only if $|R(\Delta t \lambda)| \leq 1$. If this condition is not satisfied the method will be unstable.

If a Runge Kutta method is used instead of Euler's method then the update rule can still be expressed in the form of Equation (A.8), with the polynomial given by:

$$R(z) = 1 + z \sum_j b_j + z^2 \sum_{j,k} b_j a_{jk} + z^3 \sum_{j,k,l} b_j a_{jk} a_{kl} + ...$$

The function R is called the stability function of a method and the set $S = \{z \in \mathbb{C}; |R(z)| \leq 1\}$ is called the stability domain of the method.

Figure A.1 shows the solution of Equation (A.7) with $\lambda = -10$ and two different values of Δt using the Euler's method. When $\Delta t = 0.01$ the solution converges but if this value is increased to 0.3, $\Delta t \lambda$ exits the stability domain of the method and the solution doesn't converge to the correct value. The two plots of the figure shows how the stability of the method used for solving an ODEs limits the maximum possible time step size.

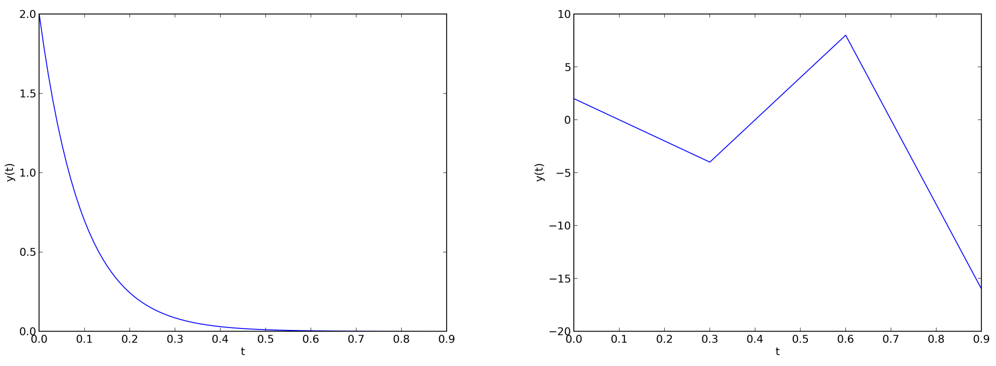


Figure A.1: Solution of Equation (A.7) for $\lambda = -10$, obtained using Euler's method. Left: $\Delta t = 0.01$. Right: $\Delta t = 0.3$

Figure A.2 shows the stability domain for Euler's method and Runge Kutta 4. For Euler's method it is a circle of radius 1 and centre -1, and for Runge Kutta 4 is a more complex shape. The plots show that the total area of the stability domain of the Runge Kutta 4 method is bigger than for Euler's method. For this reason, normally, stability will be obtained for bigger Δt if Runge Kutta 4 is used instead of Euler.

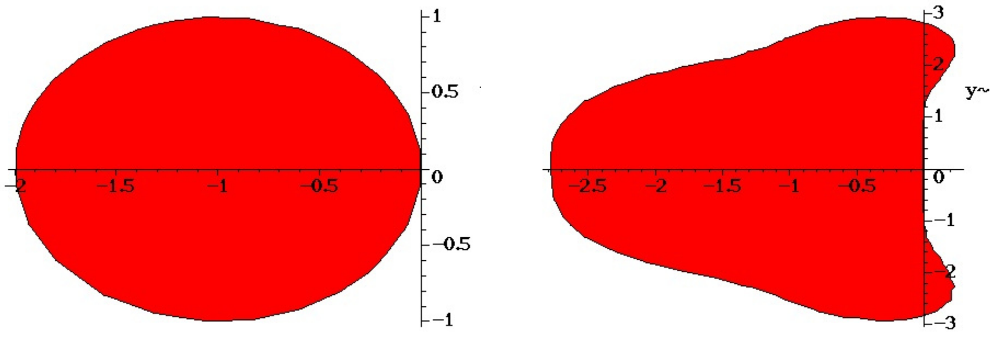


Figure A.2: Left: stability domain for Euler's method. Right: stability domain for RK4 method.

For equations more complex than (A.7) it is possible to first linearize f in the neighbourhood of a smooth solution of (A.1), $\varphi(t)$, as follows:

$$\frac{dy}{dt} = f(t, \varphi(t)) + \frac{df}{dy}(x, \varphi(t))(y(t) - \varphi(t)) + ...$$

We then define $\bar{y}(t) = y(t) - \varphi(t)$, to get:

$$\frac{d\bar{y}(t)}{dt} = \frac{df}{dy}(x, \varphi(t))\bar{y}(t) + ... = J(t)\bar{y}(t) + ...$$

To simplify the analysis we consider the Jacobian of f, $J(t)$, as constant and neglect error terms. Now if we ommit the bars and apply Euler's method to (A.1) we get the following relationship:

$$y_{k+1} = y_k + \Delta t J y_k = Rm(\Delta t J) y_k, \tag{A.9}$$

where $Rm(A)$ is a function that maps the matrix A with the matrix $1 + A$. If we assume that J is diagonizable and we can express y_0 in the basis form by the eigenvectors $v_1, ..., v_n$ of J.

$$y_0 = \sum_{i=1}^{n} \alpha_i v_i.$$

Then, this is inserted this into (A.9) we obtain:

$$y_m = \sum_{i=1}^{n} (R(\Delta t \lambda_i))^m \alpha_i v_i, \tag{A.10}$$

where the λ_i are the eigenvalues of J and $R(z)$ is the stability function of Euler's method.
Equation (A.10) will remain bounded only if for all the eigenvalues the following relationship holds:

$$|R(\Delta t \lambda_i)| \leq 1$$

Runge-Kutta methods can be used instead of Euler's method to arrive to the same relationship but with the corresponding stability function.

A.1.5 Implicit methods

The simplest implicit method is Backwards or Implicit Euler, which is similar to Euler's method presented in section A.1.2. The main difference between the two is that the rule for computing the value of the function at a new time step uses $f(t_{k+1}, y_{k+1})$ instead of $f(t_k, y_k)$ (see (A.2)). This new structure, provides an implicit equation which normally needs to be solved by an iterative method.

The method can be derived by integrating the function f between the limits of a time step:

$$y_{k+1} - y_k = \int_{t_k}^{t_{k+1}} f(t, y(t))dt,$$

and then approximating the integral with the rectangular rule:

$$\int_{t_k}^{t_{k+1}} f(t, y(t))dt \approx \Delta t f(t_{k+1}, y_{k+1}).$$

Finally, the method approximates the function by:

$$y_{k+1} = y_k + \Delta t f(t_{k+1}, y_{k+1}). \tag{A.11}$$

The method can be used for Equation (A.7) to obtain its stability domain, which is:

$$R(z) = \frac{1}{1-z}.$$

This is the exterior of the circle with radius 1 and centre 1, as shown in figure A.3. The area is much larger than those of explicit methods, as it covers almost completely the negative half-plane.

The Implicit Euler's method still has a large global truncation error of $O(\Delta t)$. More accurate implicit methods, like Runge Kutta, are also possible and a complete list is presented in [Hairer 2010]. This kind of method normally require solving more than one system of equations per time step.

A.2 Numerical method for stochastic differential equations

In this second part of the appendix we introduce the method used in this thesis for solving stochastic differential equations (SDE). In chapter 2 we describe a neural network with a system of SDE that we solve numerically either to establish

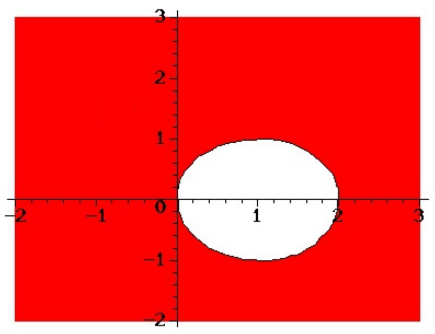


Figure A.3: Stability domain for Implicit Euler's method.

the existence of the propagation of chaos effect or to approximate the probability density function. The topics covered in this part of this appendix are essential for understanding how the SDE that are discussed in the thesis have been solved.

For more details on how to solve numerically SDE we recommend the books [Kloeden 1995] and [Mao 2007]

A.2.1 Euler-Maruyama method

We consider the following stochastic differential equation:

$$dX(t) = a(t, X(t))dt + b(t, X(t))dW(t), \qquad (A.12)$$

with initial value $X(t_0) = X_0$ and where W is a Wiener process.

The Euler-Maruyama method will produce an approximation for the SDE in (A.12) at a discrete set of values of t in a similar way as Euler's method did for ODEs (see section A.1.2). The method defines an iterative rule which creates an approximation of $X(t)$ at $t = t_1, t_2, ...$ with $(t_{k+1} - t_k) = \Delta t$. Each iteration of the method produces an approximation of X for one value of t based on the approximation obtained for the previous one. This rule is given by the following stochastic process:

$$Y_{k+1} = Y_k + a(t_k, Y_k)(\Delta t) + b(t_k, Y_k)(W_{t_{k+1}} - W_{t_k}), \qquad (A.13)$$

where Y_k is an approximation for $X(t_k)$. If $b(t_k, Y_k) = 0$ the SDE is transformed in an ODE and the method becomes the standard Euler's method described in section A.1.2.

The method may be used in a similar way to Euler's method for ODEs but now the random increments $(W_{t_{k+1}} - W_{t_k})$ need to be generated. These are random variables with mean 0 and variance Δt for which values can be generated with any random number generator capable of creating a sequences of Gaussian random numbers.

A.2.2 Error's order

SDEs have more than one possible solution starting from the same initial conditions because of the random increments. For this reason it is impossible to use the same definition of errors and convergence as in the case of ODEs. Next, we present the definition of the two different error orders that are normally used when studying numerical methods for SDE ([Kloeden 1995]).

An approximation to a stochastic process converges in the strong sense with order $\gamma \in (0, \infty]$ if there exists a constant K and a positive constant Δt_0 such that:

$$E(|X(T) - Y_T|) \leq K \Delta t^{\gamma},$$

for any $\Delta t \in (0, \Delta t_0)$. The function E represents the expected value.

An approximation to a stochastic process converges in the weak sense with order $\beta \in (0, \infty]$ if for any polynomial g there exist a finite constant K and a positive constant Δt_0 such that:

$$|E(g(X_T)) - E(g(Y_T))| \leq K \Delta t^{\beta},$$

for any $\Delta t \in (0, \Delta t_0)$.

Under appropriate condition The Euler-Maruyama method converges with weak order 1 and with strong order $\frac{1}{2}$ ([Kloeden 1995, Sauer 2012]).

Bibliography

[Abdulle 2001] A. Abdulle and A. A. Medovikov. *Second order Chebyshev methods based on orthogonal polynomials.* Numerische mathematik, vol. 90, pages 1–18, 2001. (Cited on page 97.)

[Ackermann 2001] J. Ackermann, U. Tangen, B. Bodekker, J. Breyer, E. Stoll and J. McCaskill. *Parallel random number generator for inexpensive configurable hardware cells.* Computer Physics Communications, vol. 140, pages 293–302, 2001. (Cited on page 42.)

[Amari 1972] S. Amari. *Characteristics of random nets of analog neuron-like elements.* Syst. Man Cybernet. SMC-2, 1972. (Cited on pages 18 and 19.)

[Amari 1977] S.-I. Amari. *Dynamics of pattern formation in lateral-inhibition type neural fields.* Biological Cybernetics, vol. 27, no. 2, page 77–87, 1977. (Cited on pages 18 and 19.)

[Amdahl 1967] G. Amdahl. *Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities.* AFIPS Conference Proceedings, vol. 30, pages 483–485, 1967. (Cited on page 25.)

[Angelucci 2002] A. Angelucci, J. Levitt, E. Walton, J. Hupe, J. Bullier and J. Lund. *Circuits for local and global signal integration in primary visual cortex.* The Journal of Neuroscience, vol. 22, no. 19, page 8633–8646, 2002. (Cited on page 10.)

[Anzai 1999] A. Anzai, I. Ohzawa and R. D. Freeman. *Neural Mechanisms for Processing Binocular Information I. Simple Cells.* Journal of Neurophysiology, vol. 82, pages 891–908, 1999. (Cited on page 8.)

[Baladron 2012a] J. Baladron, D. Fasoli and O. Faugeras. *Three applications of GPU computing in neuroscience.* Computing in Science and Engineering, 2012. (Cited on pages 38, 75, 136, 146 and 150.)

[Baladron 2012b] J. Baladron, D. Fasoli, O. Faugeras and J. Touboul. *Mean-field description and propagation of chaos in networks of Hodgkin-Huxley neurons.* The Journal of Mathematical Neuroscience, vol. 2, no. 1, 2012. (Cited on pages 34, 37, 38, 146 and 150.)

[Balay 1997] S. Balay, W. D. Gropp, L. C. McInnes and B. F. Smith. *Efficient Management of Parallelism in Object Oriented Numerical Software Libraries.* In E. Arge, A. M. Bruaset and H. P. Langtangen, editeurs, Modern Software Tools in Scientific Computing, pages 163–202. Birkhäuser Press, 1997. (Cited on page 74.)

[Balay 2012a] S. Balay, J. Brown, , K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith and H. Zhang. *PETSc Users Manual.* Rapport technique ANL-95/11 - Revision 3.3, Argonne National Laboratory, 2012. (Cited on page 74.)

[Balay 2012b] S. Balay, J. Brown, K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith and H. Zhang. *PETSc Web page*, 2012. http://www.mcs.anl.gov/petsc. (Cited on page 74.)

[Bastian 1994] P. Bastian and G. Wittum. *Adaptive multigrid methods: The UG concept.* Notes on numerical fluid mechanics, vol. 1, pages 17–40, 1994. (Cited on page 110.)

[Bastian 1997] P. Bastian, K. Birken, K. Johannsen, S. Lang, N. Neuß, H. Rentz-Reichert and C. Wieners. *UG – A flexible software toolbox for solving partial differential equations.* Computing and Visualization in Science, vol. 1, no. 1, pages 27–40, 1997. (Cited on page 110.)

[Battaglia 2011] D. Battaglia and D. Hansel. *Synchronous Chaos and Broad Band Gamma Rhythm in a Minimal Multi-Layer Model of Primary Visual Cortex.* PloS Computational biology, 2011. (Cited on pages 83, 85 and 94.)

[Bayoumi 2009] A. Bayoumi, M. Chu, Y. Hanafy, P. Harrell and G. Refai-Ahmed. *Scientific and engineering computing using ATI Stream technology.* Computing in Science and Engineering, vol. 11, no. 6, pages 92–97, 2009. (Cited on page 27.)

[Ben-Yishai 1995] R. Ben-Yishai, R. Bar-Or and H. Sompolinsky. *Theory of orientation tuning in visual cortex.* Proceedings of the National Academy of Sciences, vol. 92, no. 9, page 3844–3848, 1995. (Cited on pages 21 and 113.)

[Bernhard 2006] F. Bernhard and R. Keriven. *Spiking Neurons on GPUs.* In International Conference on Computational Science (4), pages 236–243, 2006. (Cited on page 30.)

[Born 2005] R. T. Born and D. C. Bradley. *Structure and Function of Visual Area MT.* Annual Review of Neuroscience, vol. 28, pages 157–189, 2005. (Cited on page 10.)

[Bosking 1997] W. Bosking, Y. Zhang, B. Schofield and D. Fitzpatrick. *Orientation Selectivity and the Arrangement of Horizontal Connections in Tree Shrew Striate Cortex.* The Journal of Neuroscience, vol. 17, no. 6, page 2112–2127, 1997. (Cited on page 113.)

[Bressloff 2000] P. Bressloff, N. Bressloff and J. Cowan. *Dynamical mechanism for sharp orientation tuning in an integrate-and-fire model of a cortical hypercolumn.* Neural computation, vol. 12, no. 11, page 2473–2511, 2000. (Cited on page 21.)

[Bressloff 2001a] P. C. Bressloff. *Traveling fronts and wave propagation failure in an inhomogeneous neural network.* Physica D: Nonlinear Phenomena, vol. 155, no. 1-2, pages 83–100, 2001. (Cited on page 20.)

[Bressloff 2001b] P. Bressloff, J. Cowan, M. Golubitsky, P. Thomas and M. Wiener. *Geometric visual hallucinations, Euclidean symmetry and the functional architecture of striate cortex.* Phil. Trans. R. Soc. Lond. B, vol. 306, no. 1407, page 299–330, 2001. (Cited on pages 9, 21, 22, 23, 113, 114 and 130.)

[Bressloff 2002a] P. C. Bressloff, J. D. Cowan, M. Golubitsky, P. J. Thomas and M. C. Wiener. *What Geometric Visual Hallucinations Tell Us about the Visual Cortex.* Neural computation, vol. 14, no. 3, pages 473–491, 2002. (Cited on pages 114 and 130.)

[Bressloff 2002b] P. Bressloff and J. Cowan. *An amplitude equation approach to contextual effects in visual cortex.* Neural computation, vol. 14, no. 3, page 493–525, 2002. (Cited on page 22.)

[Bressloff 2012] P. Bressloff. *Spatiotemporal dynamics of continuum neural fields.* Journal of Physics A: Mathematical and Theoretical, vol. 45, 2012. (Cited on page 19.)

[Brette 2007] R. Brette, M. Rudolph, T. Carnevale, M. Hines, D. Beeman, J. M. Bower, M. Diesmann, A. Morrison, P. H. Goodman, F. C. H. Jr., M. Zirpe, T. Natschläger, D. Pecevski, G. B. Ermentrout, M. Djurfeldt, A. Lansner, O. Rochel, T. Viéville, E. Muller, A. P. Davison, S. E. Boustani and A. Destexhe. *Simulation of networks of spiking neurons: a review of tools and strategies.* Journal of Computational Neuroscience, vol. 23, no. 3, page 349–398, 2007. (Cited on page 26.)

[Briggs 2000] W. L. Briggs, V. E. Henson and S. McCormick. A multigrid tutorial. Siam, 2000. (Cited on pages 98 and 110.)

[Brunel 1999] N. Brunel and V. Hakim. *Fast Global Oscillations in Networks of Integrate-and-Fire Neurons with Low Firing Rates.* Neural Computation, vol. 11, page 1621–1671, 1999. (Cited on page 19.)

[Butenhof 1997] D. R. Butenhof. Programming with posix threads. Addison-Wesley, 1997. (Cited on page 25.)

[Caceres 2011] M. J. Caceres, J. A. Carrillo and L. Tao. *A numerical solver for a nonlinear Fokker-Planck equation representation of neuronal network dynamics.* Journal of Computational Physics, vol. 230, no. 4, pages 1084–1099, 2011. (Cited on page 52.)

[Callaway 2004] E. M. Callaway. *Cell types and local circuits in primary visual cortex of the macaque monkey.* In L. M. Chalupa and J. S. Werner, editeurs, The visual neuroscience, volume 1, chapitre 42, pages 680–694. The MIT press, 2004. (Cited on page 6.)

[Chapman 2008] B. Chapman, G. Jost and R. van van der Pas. Using openmp: Portable shared memory parallel programming. The MIT press, 2008. (Cited on page 24.)

[Chavane 2011] F. Chavane, D. Sharon, D. Jancke, O. Marre, Y. Frégnac and A. Grinvald. *Lateral spread of orientation selectivity in V1 is controlled by intracortical cooperativity.* Frontiers in Systems Neuroscience, vol. 5, 2011. (Cited on pages 124, 144 and 148.)

[Chizhov 2007] A. V. Chizhov and L. J. Graham. *Population model of hippocampal pyramidal neurons, linking to refractory density approach to conductance-based neurons.* Phys. rev. E, vol. 75, no. 011924, page 114, 2007. (Cited on page 19.)

[Chossat 2009] P. Chossat and O. Faugeras. *Hyperbolic Planforms in Relation to Visual Edges and Textures Perception.* PLOS Computational biology, 2009. (Cited on pages 22 and 113.)

[Cleland 1974] B. Cleland and W. Levick. *Properties of rarely encountered types of ganglion cells in the cat's retina and on overall classification.* Journal of Physiology, vol. 240, no. 2, page 457–492, 1974. (Cited on page 4.)

[Clifford 2003] C. Clifford and M. Ibbotson. *Fundamental mechanisms of visual motion detection: models, cells and functions.* Progress in Neurobiology, vol. 68, pages 409–437, 2003. (Cited on pages 8 and 10.)

[Coddington 1998] P. D. Coddington and S.-H. Ko. *Techniques for empirical testing of parallel random number generators.* In Proc. International Conference on Supercomputing, 1998. (Cited on page 42.)

[Connor 1997] C. E. Connor, D. C. Preddie, J. L. Gallant and D. C. V. Essen. *Spatial Attention Effects in Macaque Area V4.* The Journal of Neuroscience, vol. 17, no. 9, pages 3201–3214, 1997. (Cited on page 10.)

[Coombes 2005] S. Coombes. *Waves, bumps, and patterns in neural fields theories.* Biological Cybernetics, vol. 93, no. 2, page 91–108, 2005. (Cited on pages 19 and 20.)

[Cox 1985] J. Cox, J. Ingersoll Jr and S. Ross. *A theory of the term structure of interest rates.* Econometrica: Journal of the Econometric Society, page 385–407, 1985. (Cited on page 36.)

[Cumming 2001] B. G. Cumming and G. C. DeAngelis. *The physiology of Stereopsis.* Annual review of Neuroscience, vol. 24, pages 203–238, 2001. (Cited on page 8.)

[Das 1999] A. Das and C. Gilbert. *Topography of contextual modulations mediated by short-range interactions in primary visual cortex.* Nature, vol. 399, no. 6737, page 655–661, 1999. (Cited on page 9.)

[Dayan 2001] P. Dayan and L. Abbott. Theoretical neuroscience : Computational and mathematical modeling of neural systems. MIT Press, 2001. (Cited on pages 1 and 12.)

[Dedner 2004] A. Dedner, C. Rohde, b. Schupp and M. Wesenberg. *A parallel, load balanced MHD code on locally adapted, unstructured grids in 3D.* Computing and visualization in Science, vol. 7, pages 79–96, 2004. (Cited on page 110.)

[Deng 2013] Y. Deng, P. Zhang, C. Marques, R. Powell and L. Zhang. *Analysis of Linpack and power efficiencies of the world's TOP500 supercomputers.* Parallel computing, vol. 39, pages 271–279, 2013. (Cited on page 25.)

[Destexhe 1994] A. Destexhe, Z. Mainen and T. Sejnowski. *An efficient method for computing synaptic conductances based on a kinetic model of receptor binding.* Neural Computation, vol. 6, no. 1, page 14–18, 1994. (Cited on page 18.)

[Ecker 2010] A. Ecker, P. Berens, G. Keliris, M. Bethge, N. Logothetis and A. Tolias. *Decorrelated neuronal firing in cortical microcircuits.* science, vol. 327, no. 5965, page 584, 2010. (Cited on page 37.)

[ElBoustani 2009] S. ElBoustani and A. Destexhe. *A master equation formalism for macroscopic modeling of asynchronous irregular activity states.* Neural computation, vol. 21, no. 1, page 46–100, 2009. (Cited on page 19.)

[Ermentrout 1998] B. Ermentrout. *Neural networks as spatio-temporal pattern forming systems.* Reports of Progress in Physics, vol. 61, pages 353–430, 1998. (Cited on page 20.)

[Ermentrout 2010] G. B. Ermentrout and D. Terman. Foundations of mathematical neuroscience. Springer, 2010. (Cited on pages 1, 2, 12, 17 and 18.)

[Faisal 2008] A. A. Faisal, L. P. J. Selen and D. M. Wolpert. *Noise in the nervous system.* Nature Review Neuroscience, vol. 9, pages 292–303, 2008. (Cited on page 76.)

[Faugeras 2009] O. Faugeras, J. Touboul and B. Cessac. *A constructive mean field analysis of multi population neural networks with random synaptic weights and stochastic inputs.* Frontiers in Computational Neuroscience, vol. 3, no. 1, 2009. (Cited on page 19.)

[Faye 2011] G. Faye, P. Chossat and O. Faugeras. *Analysis of a hyperbolic geometric model for visual texture perception.* The Journal of Mathematical Neuroscience, vol. 1, no. 4, 2011. (Cited on page 22.)

[Felleman 1991] D. J. Felleman and D. C. V. Essen. *Distributed hierarchical processing in the primate cerebral cortex.* Cerebral cortex, vol. 1, no. 1, pages 1–47, 1991. (Cited on page 10.)

[Felleman 1997] D. J. Felleman, A. Burkhalter and D. C. V. Essen. *Cortical Connections of Areas V3 and VP of Macaque Monkey Extrastriate Visual Cortex.* The Journal of Comparative Neurology, vol. 379, pages 21–47, 1997. (Cited on page 10.)

[Fidjeland 2009] A. K. Fidjeland, E. B. Roesch, M. P. Shanahan and W. Luk. *NeMo: A Platform for Neural Modelling of Spiking Neurons Using GPUs.* In Proceeding of the 20th IEEE International Conference on Application-specific Systems, Architectures and Processors, 2009. (Cited on page 30.)

[Field 2004] D. J. Field and A. Hayes. *Contour integration and the lateral connections of V1 neurons.* In L. M. Chalupa and J. S. Werner, editeurs, The visual neuroscience, volume 2, chapitre 70, pages 1069–1079. The MIT press, 2004. (Cited on page 130.)

[FitzHugh 1955] R. FitzHugh. *Mathematical models of threshold phenomena in the nerve membrane.* Bulletin of Mathematical Biology, vol. 17, no. 4, page 257–278, 1955. (Cited on page 14.)

[Fitzhugh 1966] R. Fitzhugh. *Theoretical Effect of Temperature on Threshold in the Hodgkin-Huxley Nerve Model.* The Journal of General Physiology, vol. 49, no. 5, page 989–1005, 1966. (Cited on page 14.)

[FitzHugh 1969] R. FitzHugh. Mathematical models of excitation and propagation in nerve, chapitre 1. H. P. Schwan, 1969. (Cited on page 14.)

[Flynn 1972] M. J. Flynn. *Some computer organizations and their effectiveness.* IEEE Transactions on Computers, vol. 21, no. 9, pages 948–960, 1972. (Cited on page 24.)

[Fourcaud 2002] N. Fourcaud and N. Brunel. *Dynamics of the Firing Probability of Noisy Integrate-and-Fire Neurons.* Neural computation, vol. 14, pages 2057–2110, 2002. (Cited on page 53.)

[Garland 2008] M. Garland, S. L. Grand, J. Nickolls, J. Anderson, J. Hardwick, S. Morton, E. Phillips, Y. Zhang and V. Volkov. *Parallel Computing Experiences with CUDA.* IEEE Micro, vol. 28, no. 4, pages 13–27, 2008. (Cited on page 27.)

[Gegenfurtner 1997] K. R. Gegenfurtner, D. C. Kiper and J. B. Levitt. *Functional Properties of Neurons in Macaque Area V3.* The Journal of Neurophysiology, vol. 77, pages 1906–1923, 1997. (Cited on page 10.)

[Geisler 2001] W. S. Geisler, J. S. Perry, B. J. Super and Gallogly. *Edge co-occurrence in natural images predicts contour grouping performance.* Vision research, vol. 41, no. 6, pages 711–724, 2001. (Cited on page 137.)

[Gerstner 1995] W. Gerstner. *Time structure of the activity in neural network models.* Physical Review E, vol. 51, no. 1, page 738–758, 1995. (Cited on page 18.)

[Gerstner 2002] W. Gerstner and W. M. Kistler. Spiking neuron models. Cambridge University Press, 2002. (Cited on pages 1, 12, 13, 16, 17 and 76.)

[Gewaltig 2007] M.-O. Gewaltig and M. Diesmann. *NEST (NEural Simulation Tool).* Scholarpedia, vol. 2, no. 4, page 1430, 2007. (Cited on page 26.)

[Glaskowsky 2009] P. N. Glaskowsky. *NVIDIA's Fermi: The first complete GPU Computing Architecture.* NVIDIA white paper, 2009. (Cited on pages 27, 28 and 29.)

[Goldwyn 2011] J. H. Goldwyn, N. S. Imennov, M. Famulare and E. Shea-Brown. *Stochastic differential equation models for ion channel noise in Hodgkin-Huxley neurons.* Physical Review E, vol. 83, 2011. (Cited on page 34.)

[Golomb 1997] D. Golomb and Y. Amitai. *Propagating Neuronal Discharges in Neocortical Slices: Computational and Experimental Study.* Journal of Neurophysiology, vol. 78, pages 1199–1211, 1997. (Cited on page 20.)

[Gropp 1994] W. Gropp, E. Lusk and A. Skjellum. Using mpi : portable parallel programming with the message-passing interface. The MIT press, 1994. (Cited on page 25.)

[Gross 2002] C. G. Gross. *Genealogy of the "Grandmother Cell".* Neuroscientist, vol. 8, no. 5, pages 512–518, 2002. (Cited on page 92.)

[Guo 2005a] Y. Guo and C. C. Chow. *Existence and Stability of Standing Pulses in Neural Networks: I. Existence.* SIAM Journal on Applied Dynamical Systems, vol. 4, no. 2, page 217–248, 2005. (Cited on page 20.)

[Guo 2005b] Y. Guo and C. C. Chow. *Existence and Stability of Standing Pulses in Neural Networks: II Stability.* SIAM Journal on Applied Dynamical Systems, vol. 4, page 249–281, 2005. (Cited on page 20.)

[Gutkin 2002] B. S. Gutkin, C. R. Laing, B. Ermentrout and W. C. Troy. *Multiple Bumps in a Neuronal Model of Working Memory.* SIAM Journal on Applied Mathematics, vol. 63, no. 1, pages 62–97, 2002. (Cited on page 20.)

[Hairer 2008] E. Hairer and G. Wanner. Solving ordinary differential equations i: Nonstiff problems. Springer, 2008. (Cited on page 151.)

[Hairer 2010] E. Hairer and G. Wanner. Solving ordinary differential equations ii: stiff and differential-algebraic problems. Springer, 2010. (Cited on pages 95, 151 and 156.)

[Hamker 2006] F. H. Hamker and M. Zirnsak. *V4 receptive field dynamics as predicted by a systems-level model of visual attention using feedback from the frontal eye field.* Neural Networks, vol. 19, pages 1371–1382, 2006. (Cited on page 10.)

[Hamker 2008] F. H. Hamker, M. Zirnsak, D. Calow and M. Lappe. *The Peri-Saccadic Perception of Objects and Space.* Plos computational biology, vol. 4, no. 2, 2008. (Cited on page 10.)

[Hansel 1997] D. Hansel and H. Sompolinsky. *Modeling feature selectivity in local cortical circuits.* In C. Koch and I. Segev, editeurs, Methods of neuronal modeling, chapitre 13, page 499–567. The MIT press, 1997. (Cited on pages 9 and 20.)

[Hegde 2000] J. Hegde and D. V. Essen. *Selectivity for complex shapes in primate visual area V2.* Journal of Neuroscience, vol. 20, 2000. (Cited on page 10.)

[Hennessy 2007] J. L. Hennessy and D. A. Patterson. Computer architecture: a quantitative approach. Morgan Kaufmann and Elsevier, 2007. (Cited on page 24.)

[Hines 2002] M. Hines and N. Carnevale. *The NEURON Simulation Environment.* In The Handbook of Brain Theory and Neural Networks, pages 769–773. The MIT press, 2002. (Cited on page 26.)

[Hines 2008] M. L. Hines, H. Markram and F. Schurmann. *Fully implicit parallel simulation of single neurons.* Journal of Computational Neuroscience, vol. 25, no. 3, pages 439–448, 2008. (Cited on page 26.)

[Hoch 2003] T. Hoch, G. Wenning and K. Obermayer. *Optimal noise-aided signal transmission through populations of neurons.* Phys. Rev. E, vol. 68, page 011911, 2003. (Cited on page 76.)

[Hodgkin 1952] A. Hodgkin and A. Huxley. *A quantitative description of membrane current and its application to conduction and excitation in nerve.* Journal of Physiology, vol. 117, page 500–544, 1952. (Cited on page 11.)

[Hubel 1962] D. Hubel and T. Wiesel. *Receptive fields, binocular interaction and functional architecture in the cat visual cortex.* J Physiol, vol. 160, page 106–154, 1962. (Cited on pages 6 and 7.)

[Hubel 1977] D. Hubel and T. Wiesel. *Functional architecture of macaque monkey visual cortex.* Proceedings of the Royal Society, London [B], page 1–59, 1977. (Cited on pages 8 and 9.)

[Hubel 1995] D. Hubel. Eye, brain and vision. W. H. Freeman, 1995. (Cited on pages 2 and 4.)

[Issa 2000] N. P. Issa, C. Trepel and M. P. Stryker. *Spatial Frequency Maps in Cat Visual Cortex.* The Journal of Neuroscience, vol. 20, no. 22, pages 8504–8514, 2000. (Cited on page 9.)

[Izhikevich 2006] E. M. Izhikevich and R. FitzHugh. *FitzHugh-Nagumo model.* Scholarpedia, vol. 1, no. 9, page 1349, 2006. (Cited on page 14.)

[Izhikevich 2007] E. Izhikevich. Dynamical systems in neuroscience: The geometry of excitability and bursting. MIT Press, 2007. (Cited on pages iv, vi, 2, 12, 14, 15 and 68.)

[Jeng 2000] C. Jeng, K. Tan and J. A. R. Blais. *PLFG: A Highly Scalable Parallel Pseudo-random Number Generator for Monte Carlo Simulations.* High Performance Computing and Networking, pages 127–135, 2000. (Cited on page 42.)

[Jones 1987] J. P. Jones and L. A. Palmer. *An evaluation of the two-dimensional Gabor filter model of simple receptive fields in cat striate cortex.* Journal of neurophysiology, vol. 58, no. 6, pages 1233–1258, 1987. (Cited on page 131.)

[Kandel 2000] E. R. Kandel, J. H. Schwartz and T. M. Jesell. Principles of neural science. Mc Graw Hill, 2000. (Cited on page 1.)

[Kaplan 2004] E. Kaplan. *The M,P and K pathways to the primate visual system.* In L. M. Chalupa and J. S. Werner, editeurs, The visual neuroscience, volume 1, chapitre 30, pages 481–493. The MIT press, 2004. (Cited on page 5.)

[Khan 1999] I. R. Khan and R. Ohba. *Closed-form expressions for the finite dfference approximations of first and higher derivatives based on Taylor series.* Journal of Computational and Applied Mathematics, vol. 107, pages 179–193, 1999. (Cited on page 47.)

[Kilpatrick 2008] Z. P. Kilpatrick, S. E. Folias and P. C. Bressloff. *Traveling Pulses and Wave Propagation Failure in Inhomogeneous Neural Media.* SIAM Journal on Applied Dynamical Systems, vol. 7, no. 1, pages 161–185, 2008. (Cited on page 20.)

[Kindratenko 2009] V. V. Kindratenko, J. J. Enos, G. Shi, M. T. Showerman, G. W. Arnold, J. E. Stone, J. C. Phillips and W. mei Hwu. *GPU Clusters for High-Performance Computing.* In Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on, 2009. (Cited on page 27.)

[Kirk 2010] D. B. Kirk and W. mei W. Hwu. Programming massively parallel processors. Morgan Kaufman Publishers, 2010. (Cited on page 30.)

[Kloeden 1995] P. E. Kloeden and E. Platen. Numerical solution to stochastic differential equations. Springer, 1995. (Cited on pages 157 and 158.)

[Knoblauch 2005] A. Knoblauch and G. Palm. *What is signal and what is noise in the brain?* Biosystems, vol. 79, no. 1–3, pages 83 – 90, 2005. (Cited on page 76.)

[Knoll 2004] D. Knoll and D. Keyes. *Jacobian-free Newton–Krylov methods: a survey of approaches and applications.* Journal of Computational Physics, vol. 193, pages 357–397, 2004. (Cited on page 97.)

[Kuck 2011] D. J. Kuck. *Parallel computing.* In Encyclopedia of Parallel Computing, pages 1409–1416. Springer, 2011. (Cited on pages 24 and 25.)

[Kyriazi 1993] H. T. Kyriazi and D. J. Simons. *Thalamocortical response transformations in simulated whisker barrels.* The journal of neuroscience, vol. 13, no. 4, pages 1601–1615, 1993. (Cited on page 80.)

[Laing 2003a] C. R. Laing and W. C. Troy. *PDE Methods for Nonlocal Models.* SIAM Journal on Applied Dynamical Systems, vol. 2, no. 3, pages 487–516, 2003. (Cited on page 20.)

[Laing 2003b] C. R. Laing and W. C. Troy. *Two-bump solutions of Amari-type models of neuronal pattern formation.* Physica D Nonlinear Phenomena, vol. 178, no. 3-4, pages 190–218, 2003. (Cited on page 20.)

[Lecar 2007] H. Lecar. *Morris-Lecar model.* Scholarpedia, vol. 2, no. 10, page 1333, 2007. (Cited on page 15.)

[Lee 2005] S.-H. Lee, R. Blake and D. J. Heeger. *Travelling waves of activity in primary visual cortex during binocular rivalry.* Nature neuroscience, vol. 8, no. 1, pages 22–23, 2005. (Cited on page 20.)

[Lund 2003] J. S. Lund, A. Angelucci and P. C. Bressloff. *Anatomical Substrates for Functional Columns in Macaque Monkey Primary Visual Cortex.* Cerebral Cortex, vol. 12, page 15–24, 2003. (Cited on pages 10 and 113.)

[Lundstrom 2003] M. Lundstrom. *Moore's Law Forever?* Science, vol. 299, no. 5604, pages 210–211, 2003. (Cited on page 24.)

[Ma 2006] W. J. Ma, J. M. Beck, P. E. Latham and A. Pouget. *Bayesian inference with probabilistic population codes.* Nature Neuroscience, vol. 9, pages 1432–1438, 2006. (Cited on page 92.)

[Makoto 1998] M. Makoto and T. Nishimura. *Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator.* ACM Transactions on Modeling and Computer Simulation, vol. 8, no. 1, pages 1–30, 1998. (Cited on page 42.)

[Mao 2007] X. Mao. Stochastic differential equations and applications. Horwood, 2007. 2nd Edition. (Cited on pages 35, 41 and 157.)

[Masland 2001a] R. Masland. *Neuronal diversity in the retina*. Current Opinion in Neurobiology, vol. 11, no. 4, page 431–436, 2001. (Cited on page 4.)

[Masland 2001b] R. H. Masland. *The fundamental plan of the retina*. Nature Neuroscience, vol. 4, pages 877–886, 2001. (Cited on page 4.)

[Mattia 2002] M. Mattia and P. Del Giudice. *Population dynamics of interacting spiking neurons*. Physical Review E, vol. 66, no. 5, page 51917, 2002. (Cited on page 19.)

[Mattson 2004] T. G. Mattson, B. A. Sanders and B. L. Massingill. Patterns for parallel programming. Addison Wesley, 2004. (Cited on page 24.)

[McAdams 1999] C. J. McAdams and J. H. R. Maunsell. *Effect of attention on orientation-tuning functions of single neurons in Macaque cortical area V4*. The Journal of Neuroscience, vol. 19, no. 1, pages 431–441, 1999. (Cited on page 10.)

[Michea 2010] D. Michea and D. Komatitsch. *Accelerating a 3D finite-difference wave propagation code using GPU graphics cards*. Geophysics Journal, vol. 182, pages 389–402, 2010. (Cited on page 50.)

[Micikevicius 2009] P. Micikevicius. *3D finite difference computation on GPUs using CUDA*. In Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units, pages 79–84, 2009. (Cited on page 50.)

[Mohammad A. Bhuiyan 2010] V. K. P. Mohammad A. Bhuiyan and M. C. Smith. *Acceleration of spiking neural networks in emerging multi-core and GPU architectures*. In 2010 IEEE International Symposium of Parallel and Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010. (Cited on page 30.)

[Morris 1981] C. Morris and H. Lecar. *Voltage Ooscillations in the Barnacle giant muscle fiber*. Biophysical journal, vol. 35, no. 1, pages 193–213, 1981. (Cited on page 15.)

[Morrison 2005] A. Morrison, C. Mehring, T. Geisel, A. Aertsen and M. Diesmann. *Advancing the boundaries of high-connectivity network simulation with distributed computing*. Neural computation, vol. 17, no. 8, pages 1776–1801, 2005. (Cited on page 26.)

[Morton 2005] K. Morton and D. Mayers. Numerical solution of partial differential equations: an introduction. Cambridge Univ Press, 2005. (Cited on pages 47 and 97.)

[Mountcastle 1997] V. B. Mountcastle. *The columnar organization of the neocortex*. Brain, vol. 120, pages 701–722, 1997. (Cited on page 94.)

[Nageswaran 2009] J. M. Nageswaran, N. Dutt, J. L. Krichmar, A. Nicolau and A. Veidenbaum. *Efficient simulation of large-scale spiking neural networks using CUDA graphics processors.* In IJCNN'09 Proceedings of the 2009 international joint conference on Neural Networks, 2009. (Cited on page 30.)

[Nelson 2004] R. Nelson and H. Kolb. *ON and OFF pathways in the vertebrate retina and visual system.* In L. M. Chalupa and J. S. Werner, editeurs, The visual neuroscience, volume 1, chapitre 18, pages 260–278. The MIT press, 2004. (Cited on page 4.)

[Nickolls 2010] J. Nickolls and W. J. Dally. *The GPU computing era.* IEEE Micro, vol. 30, no. 2, pages 56–69, 2010. (Cited on page 27.)

[Obermayer 1993] K. Obermayer and G. G. Blasdel. *Geometry of orientation and ocular dominance columns in monkey striate cortex.* The Journal of Neuroscience, vol. 13, no. 10, pages 4114–4129, 1993. (Cited on page 8.)

[Ohki 2006] K. Ohki, S. Chung, P. Kara, M. Hubener, T. Bonhoeffer and R. Reid. *Highly ordered arrangement of single neurons in orientation pinwheels.* Nature, vol. 442, no. 7105, page 925–928, 2006. (Cited on page 6.)

[Ohzawa 1990] I. Ohzawa, G. C. DeAngelis and R. D. Freeman. *Stereoscopic depth discrimination in the visual cortex: neurons ideally suited as disparity selectors.* Science, vol. 249, no. 4972, pages 1037–1041, 1990. (Cited on page 8.)

[Okamoto 2011] T. Okamoto, K. Ikezoe, H. T. M. Watanabe, K. Aihara and I. Fujita. *Predicted contextual modulation varies with distance from pinwheel centers in the orientation preference map.* Scientific reports, 2011. (Cited on pages 6 and 7.)

[Owens 2008] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone and J. C. Phillips. *GPU Computing.* Proceedings of the IEEE, vol. 96, no. 5, pages 879–899, 2008. (Cited on page 27.)

[Pakdaman 2010] K. Pakdaman, M. Thieullen and G. Wainrib. *Fluid limit theorems for stochastic hybrid systems with application to neuron models.* Advances in Applied Probability, vol. 42, no. 3, page 761–794, 2010. (Cited on page 34.)

[Parent 1989] P. Parent and S. Zucker. *Trace inference, curvature consistency, and curve detection.* IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 11, no. 8, page 823–839, 1989. (Cited on page 137.)

[Peinado 2000] A. Peinado. *Traveling Slow Waves of Neural Activity: A Novel Form of Network Activity in Developing Neocortex.* The Journal of Neuroscience, vol. 20, pages 1–6, 2000. (Cited on page 20.)

[Petersen 2007] C. C. Petersen. *The Functional Organization of the Barrel Cortex.* Neuron, vol. 56, pages 339–355, 2007. (Cited on page 80.)

[Pinto 1996] D. J. Pinto, J. C. Brumberg, D. J. Simons and G. B. Ermentrout. *A quantitative population model of whisker barrels: re-examining the Wilson-Cowan equations.* Journal of computational neuroscience, vol. 3, no. 3, pages 247–264, 1996. (Cited on page 80.)

[Pinto 2000] D. J. Pinto, J. C. Brumberg and D. J. Simons. *Circuit dynamics and coding strategies in rodent somatosensory cortex.* Jounal of neurophysiology, vol. 83, no. 3, pages 1158–1166, 2000. (Cited on pages 80 and 81.)

[Pinto 2003] D. J. Pinto, J. A. Hartings, J. C. Brumberg and D. J. Simons. *Cortical Damping: Analysis of Thalamocortical Response Transformations in Rodent Barrel Cortex.* Cerebral Cortex, vol. 13, pages 33–44, 2003. (Cited on page 80.)

[Pouget 2000] A. Pouget, P. Dayan and R. Zemel. *Information Processing with population codes.* Nature Review Neuroscience, vol. 1, no. 2, pages 125–132, 2000. (Cited on page 92.)

[Roe 1986] P. Roe. *Characteristic-Based Schemes for the Euler Equations.* Annual Review of Fluid Mechanics, vol. 18, pages 337–365, 1986. (Cited on page 52.)

[Rolls 2002] E. T. Rolls and G. Deco. Computational neuroscience of vision. Oxford university press, 2002. (Cited on page 83.)

[Rolls 2010] E. Rolls and G. Deco. The noisy brain: stochastic dynamics as a principle of brain function. Oxford university press, 2010. (Cited on pages 53 and 83.)

[Rubin 2004] J. E. Rubin and W. C. Troy. *Sustained Spatial Patterns of Activity in Neuronal Populations without Recurrent Excitation.* SIAM Journal on Applied Mathematics, vol. 64, no. 5, pages 1609–1635, 2004. (Cited on page 20.)

[Rust 2005] N. C. Rust, O. Schwartz, J. A. Movshon and eero P. Simoncelli. *Spatiotemporal Elements of Macaque V1 Receptive Fields.* Neuron, vol. 46, pages 945–956, 2005. (Cited on page 6.)

[Sanguinetti 2010] G. Sanguinetti, G. Citti and A. Sarti. *A model of natural image edge co-occurrence in the rototranslation group.* Journal of vision, vol. 10, no. 14, 2010. (Cited on page 137.)

[Sauer 2012] T. Sauer. *Numerical solution of stochastic differential equations in finance.* In J.-C. Duan, W. K. Harde and J. E. Gentle, editeurs, Handbook of computational finance, pages 529–550. Springer, 2012. (Cited on page 158.)

[Schiesser 1991] W. Schiesser. The numerical method of lines: Integration of partial differential equations. Academic Press, San Diego, 1991. (Cited on page 47.)

[Shadlen 1998] M. N. Shadlen and W. T. Newsome. *The Variable Discharge of Cortical Neurons: Implications for Connectivity, Computation, and Information Coding.* The Journal of Neuroscience, vol. 18, no. 10, pages 3870–3896, 1998. (Cited on page 76.)

[Sherman 1996] S. M. Sherman and R. W. Guillery. *Functional organization of thalamocortical relays.* Journal of Neurophysiology, vol. 76, no. 3, pages 1367–1395, 1996. (Cited on page 5.)

[Sherman 2004] S. M. Sherman and R. Guillery. *The visual relays in the thalamus.* In L. M. Chalupa and J. S. Werner, editeurs, The visual neuroscience, volume 1, chapitre 35, pages 565–591. The MIT press, 2004. (Cited on page 5.)

[Sincich 2002] L. C. Sincich and J. C. Horton. *Divided by Cytochrome Oxidase: A Map of the Projections from V1 to V2 in Macaques.* Science, pages 1734–1737, 2002. (Cited on page 10.)

[Sisson 2006] B. R. Sisson, A. Robert and L. Cayton. The american midwest: An interpretive encyclopedia (midwestern history and culture), pages 1487–1489. Indiana University Press, 2006. (Cited on page 24.)

[Slovin 2002] H. Slovin, A. Arieli, R. Hildesheim and A. Grinvald. *Long-Term Voltage-Sensitive Dye Imaging Reveals Cortical Dynamics in Behaving Monkeys.* Journal of Neurophysiology, vol. 88, pages 3421–3438, 2002. (Cited on page 6.)

[Sommeijer 1997] B. Sommeijer, L. Shampine and J. Verwer. *RKC: An explicit solver for parabolic PDEs.* Journal of computational and applied mathematics, vol. 88, pages 315–326, 1997. (Cited on page 98.)

[Srinivasana 2003] A. Srinivasana, M. Mascagnib and D. Ceperleyc. *Testing parallelrandomnumber generators.* Parallel computing, vol. 29, no. 1, pages 69–94, 2003. (Cited on page 41.)

[Stone 2010] J. E. Stone, D. Gohara and G. Shi. *OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems.* Computing in Science and Engineering, vol. 12, no. 3, pages 66–73, 2010. (Cited on page 27.)

[Strikwerda 2004] J. C. Strikwerda. Finite difference schemes and partial differential equations. SIAM, 2004. (Cited on page 47.)

[Sulaiman 2009] N. Sulaiman, Z. A. Obaid, M. H. Marhaban and M. N. Hamidon. *Design and Implementation of FPGA-Based Systems-A Review.* Australian Journal of Basic and Applied Sciences, vol. 3, no. 4, pages 3575–3596, 2009. (Cited on page 25.)

[Thorpe 1996] S. Thorpe, D. Fize and C. Marlot. *Speed of processing in the human visual system.* Nature, vol. 381, page 520–522, 1996. (Cited on page 76.)

[Touboul 2012] J. Touboul, G. Hermann and O. Faugeras. *Noise-induced behaviors in neural mean field dynamics.* SIAM Journal on Applied dynamical Systems, vol. 11, no. 1, page 49–81, 2012. (Cited on page 53.)

[Tritsiklis 1989] J. N. Tritsiklis. *A comparisson of Jacobi and Gauss-Seidel parallel iterations.* Applied mathematics letters, vol. 2, no. 2, pages 167–170, 1989. (Cited on page 98.)

[Trottenberg 2001] U. Trottenberg, C. Oosterlee and A. Schuller. Multigrid. Academic press, 2001. (Cited on pages 98, 99 and 110.)

[Ts'o 1990] D. Y. Ts'o, R. D. Frostig, E. E. Lieke and A. Grinvald. *Functional organization of primate visual cortex revealed by high resolution optical imaging.* Science, 1990. (Cited on page 6.)

[Tucker 2004] T. R. Tucker and D. Fitzpatrick. *Contributions of vertical and horizontal circuits to the responde properties of neurons in primary visual cortex.* In L. M. Chalupa and J. S. Werner, editeurs, The visual neuroscience, volume 1, chapitre 46, pages 733–746. The MIT press, 2004. (Cited on page 10.)

[Ungerleider 2008] L. G. Ungerleider, T. W. Galkin, R. Desimone and R. Gattass. *Cortical Connections of Area V4 in the Macaque.* Cerebral Cortex, vol. 18, pages 477–499, 2008. (Cited on page 10.)

[van Der Houwen 1983] P. van Der Houwen and B. P. Sommeijer. *On the Internal Stability of Explicit, m-Stage Runge-Kutta Methods for Large m-Values.* ZAMM, vol. 60, no. 10, pages 479–485, 1983. (Cited on page 97.)

[Veltz 2011] R. Veltz. *Nonlinear analysis methods in neural field models.* PhD thesis, University of Paris Est, 2011. (Cited on pages 21, 113, 114, 115, 141, 144 and 148.)

[Verwer 1990] J. Verwer, W. Hundsdorfer and B. Sommeijer. *Convergence properties of the Runge-Kutta-Chebyshev method.* Numerische mathematik, vol. 57, pages 157–178, 1990. (Cited on page 97.)

[Verwer 1996] J. Verwer. *Explicit Runge-Kutta methods for parabolic partial differential equations.* Applied numerical mathematics, vol. 22, pages 359–379, 1996. (Cited on page 97.)

[Wainrib 2010] G. Wainrib. *Randomness in Neurons: a multiscale probabilistic analysis.* PhD thesis, Ecole Polytechnique, 2010. (Cited on page 34.)

[Wassle 2004] H. Wassle. *Parallel processing in the mammalian retina.* Nature Reviews Neuroscience, vol. 5, no. 10, pages 747–757, 2004. (Cited on page 4.)

[Weliky 1996] M. Weliky, W. H. Bosking and D. Fitzpatrick. *A systematic map of direction preference in primary visual cortex.* Nature, vol. 379, pages 725–728, 1996. (Cited on page 9.)

[Wilson 1972] H. Wilson and J. Cowan. *Excitatory and inhibitory interactions in localized populations of model neurons.* Biophys. J., vol. 12, page 1–24, 1972. (Cited on pages 18 and 19.)

[Wilson 1973] H. Wilson and J. Cowan. *A mathematical theory of the functional dynamics of cortical and thalamic nervous tissue.* Biological Cybernetics, vol. 13, no. 2, page 55–80, 1973. (Cited on pages 18 and 19.)

[Wittenbrink 2011] C. M. Wittenbrink, E. Kilgariff and A. Prabhu. *Fermi GF100 GPU Architecture.* IEEE Micro, vol. 31, no. 2, pages 50–59, 2011. (Cited on page 27.)

[Wohrer 2008] A. Wohrer. *The vertebrate retina: a functional review.* Rapport technique 6532, INRIA, 2008. (Cited on page 4.)

[Xu 2001] X. Xu, J. M. Ichida, J. D. Allison, J. D. Boyd, A. B. Bonds and V. Casagrande. *A comparison of koniocellular, magnocellular and parvocellular receptive field properties in the lateral geniculate nucleus of the owl monkey (Aotus trivirgatus).* Journal of Physiology, vol. 531, no. Pt. 1, pages 203–218, 2001. (Cited on page 5.)

[Zemel 1998] R. S. Zemel, P. Dayan and A. Pouget. *Probabilistic interpretation of population codes.* Nature Review Neuroscience, vol. 10, pages 403–430, 1998. (Cited on page 92.)

[Zirnsak 2010] M. Zirnsak, M. Lappe and F. H. Hamker. *The spatial distribution of receptive field changes in a model of peri-saccadic perception: Predictive remapping and shifts towards the saccade target.* Vision Research, vol. 50, pages 1328–1337, 2010. (Cited on page 10.)