University of Nottingham

School of Mathematical Sciences

The Numerical Solution of Neural Field Models Posed on Realistic Cortical Domains

Sammy Joseph Petros

A thesis submitted to the University of Nottingham for the degree of

Doctor of Philosophy

January 2023

For those who are no longer here to see me complete my journey

Abstract

The mathematical modelling of neural activity is a hugely complex and prominent area of exploration that has been the focus of many researchers since the mid 1900s. Although many advancements and scientific breakthroughs have been made, there is still a great deal that is not yet understood about the brain.

There have been a considerable amount of studies in mathematical neuroscience that consider the brain as a simple one-dimensional or two-dimensional domain; however, this is not biologically realistic and is primarily selected as the domain of choice to aid analytical progress. The primary aim of this thesis is to develop and provide a novel suite of codes to facilitate the computationally efficient numerical solution of large-scale delay differential equations, and utilise this to explore both neural mass and neural field models with space-dependent delays. Through this, we seek to widen the scope of models of neural activity by posing them on realistic cortical domains and incorporating real brain data to describe non-local cortical connections. The suite is validated using a selection of examples that compare numerical and analytical results, along with recreating existing results from the literature. The relationship between structural connectivity and functional connectivity is then analysed as we use an eigenmode fitting approach to inform the desired stability regimes of a selection of neural mass models with delays. Here, we explore a next-generation neural mass model developed by Coombes and Byrne [36], and compare results to the more traditional Wilson-Cowan formulation [180, 181]. Finally, we examine a variety of solutions to three different neural field models that incorporate real structural connectivity, path length, and geometric surface data, using our NFESOLVE library to efficiently compute the numerical solutions. We demonstrate how the

field version of the next-generation model can yield intricate and detailed solutions which push us closer to recreating observed brain dynamics.

Acknowledgements

First and foremost I would like to thank my fantastic supervisors who have been there for me throughout the PhD. I have had many ups and downs, but you were always there to support me and talk through any worries or concerns that I had. To Dr. Daniele Avitabile, thank you for making my first two years so enjoyable. I will be forever grateful for your time and expertise, in both mathematical and pastoral capacities. To Prof. Stephen Coombes, thank you for taking over as my principal supervisor and for supporting me throughout. You have consistently helped get me through the challenges that I have faced, and never turned me away from your door when I needed help. Finally, to Prof. Paul Houston and Prof. Stamatios Sotiropoulos, you have both given me a great deal of your time and I am truly thankful to have received the supervision and support you offered me.

I would also like to thank my assessors Dr Reuben O'Dea and Dr Kyle Wedgwood for conducting my viva and giving me constructive feedback on my work.

A huge thank you to all my friends who have filled my time at Nottingham with laughter. The endless common room discussions and stories will always be remembered. A special shout out to Sunil for being a great friend and always helping me with my numerous mathematical questions. Last but not least, I would like to thank Abigail. We started the PhD journey together and have been through every step of the way together. I will always be eternally grateful for your friendship and our innate ability to keep each other motivated when we have felt like we are having a hard time.

My family deserve a great deal of thanks for the endless phone calls

and complaining that they have endured. Without your support I have no doubt that I would not have seen things through to this point. I especially would like to acknowledge Mum and Ma for making my morning walks to the office so enjoyable and putting me in a good mood for each and every day. Finally, I would like to thank Aimee. You have had to put up with me at my most stressed and grumpy, and I will never forget the numerous times that you have reassured and motivated me to keep going and get to the end. Even through your pregnancy with Oscar, you have still always been there for me and supported me to the best of your ability.

Publications

Michael Forrester, Sammy Petros, Yi Ming Lai, Reuben D O'Dea, Stephen Coombes, Stamatios Sotiropoulos. Whole brain functional connectivity: insights from next generation neural mass modelling incorporating electrical synapses. *PLoS Computational Biology*, in preparation

Conferences Attended

- Winter School on Deterministic and Stochastic Models in Neuroscience. Institut de Mathématiques de Toulouse, France. December 2017.
- International Conference on Mathematical Neuroscience (ICMNS). Antibes Juan-Les-Pins, France. June 2018.
- Summer School on Nonlinear Dynamics in Life Sciences with Applications to Neuroscience and Psychology. McGill University, Canada. June 2018.
- British Applied Mathematics Colloquium (BAMC). University of Bath, United Kingdom. April 2019.
- International Conference on Mathematical Neuroscience (ICMNS). University of Copenhagen, Denmark. June 2019.
- Threshold Networks. University of Nottingham, United Kingdom. July 2019.

Posters

- "The Numerical Solution of Neural Field Models Posted on Realistic Cortical Domains". British Applied Mathematics Colloquium (BAMC). April 2019.

- "The Numerical Solution of Neural Field Models Posted on Realistic Cortical Domains". International Conference on Mathematical Neuroscience (ICMNS). June 2019

- "The Numerical Solution of Neural Field Models Posted on Realistic Cortical Domains". Threshold Networks. July 2019

CONTENTS

1

INTRODUCTION

> *"There are billions of neurons in our brains, but what are neurons? Just cells. The brain has no knowledge until connections are made between neurons. All that we know, all that we are, comes from the way our neurons are connected."*
>
> - Tim Berners-Lee

The brain is one of the most complex natural structures that has ever existed. Scientific advancements in studying cognition and brain function have developed phenomenally over the last century; however, there is still so much that is not yet understood. There is an enormous ethical quandary around performing experiments on living organisms, leading to the question of whether the brain can truly ever be understood via non-invasive and morally responsible methods. This is where the area of mathematical neuroscience was born. If brain phenomena can be modelled and accurately recreated using mathematics, the potential applications are endless. From curing brain-related diseases to artificially creating cognitive reasoning, there are numerous domains which could benefit from understanding how the brain works.

This thesis gives an overview of some of the mathematical neuroscience techniques and applications that have been studied by researchers to date, along with discussing how models of neural activity are constructed, and the numerical methods required to compute solutions for them. The

primary aim of this thesis is to provide a suite of numerical differential equation solvers to evolve large-scale models of neural activity, posed on realistic cortical domains, with the inclusion of space-dependent delays. We explore multiple different settings for these models and demonstrate how modern computing technology can be used to efficiently simulate cortical dynamics at the macroscopic level. Incorporating real brain data in the form of structural connectivity, path length, and cortical mesh data, we simulate both neural mass and neural field models, and demonstrate how our suite of solvers can be utilised to yield interesting and novel results in these areas.

Chapter 2 begins by setting the scene and introducing the relevant biological background knowledge required to understand how the brain works at the cellular level. We explore several key models of various neural phenomena which have paved the way for the modern formulations that we consider later on in this work. This includes explanations of how axonal delays can be included to give a more biologically realistic interpretation of neuronal interactions. Next, we give a description of the medical imaging techniques used to collect and process the brain data that we incorporate into models of neural activity with the aim of pushing simulations closer to recreating observed brain function. Finally, we introduce numerical quadrature methods for approximating integrals, with a specific focus on Gauss-Legendre quadrature, to pave the way for exploring the numerical schemes required to solve neural field models.

Following on from this, Chapter 3 discusses an array of analytical and numerical techniques used in the world of neural modelling. We start by detailing the construction of an arbitrary non-linear network, along with linear stability analysis for the network around a homogeneous steady state, in order to yield general results which can be applied to the neural mass models that we explore later in Chapter 6. The next section of this chapter explores the numerical methods required to spatially discretise neural field models in order to construct a system of ODEs that can then

be evolved temporally using numerical differential equation solution algorithms. This begins by illustrating how a discrete spatial mesh is constructed from a continuum domain, before moving on to showing how numerical integration techniques can be applied to approximate the integral term found in neural field models. Combining these two processes, we then demonstrate the complete spatial discretisation of the standard neural field equation. Lastly, we perform an error convergence analysis to show that the discretisation methods yield results that conform to theoretical expectations.

After presenting the main analytical and numerical techniques that we make use of across this thesis, we move on to Chapter 4. Here, we showcase the design of the NFESOLVE library: a suite of codes designed to efficiently and effectively solve large-scale models of neural activity with multiple delays. We begin by detailing the choices made for the tools used to build the library, before breaking down each section of the code and a giving a detailed explanation of the implementation. First, we give a description of the mesh handler module. This contains everything to do with storing and manipulating the properties of a geometric mesh, along with a quadrature library for performing numerical integration across any given mesh. Following this, we break down the design of the differential equation solver sub-packages, beginning with the ODE solution suite and subsequently moving on to the DDE solution suite. This then brings us to the main area of our code that improves upon existing solvers: the sparse DDE solver module. Here, we detail how our sparse solvers remove the unnecessary computational burden created by performing calculations that are never actually required in the solution process. Finally, we outline how the code can be parallelised to improve efficiency when dealing with systems containing a large number of delays.

An important part of any sort of software development is validating that the code performs to the desired expectations. Chapter 5 takes the NFESOLVE library and puts it through its paces to validate that each of the solvers

conform to their expected rates of convergence, along with confirming that the library can be used to efficiently and correctly solve equations in the form of the delayed neural field models that we are interested in exploring. For the latter, we begin by exploring how the standard neural field model behaves when the spatial domain of choice is a sphere. This paves the way for the work seen later in Chapter 7, as a spherical domain can be viewed as being a step in between a traditional two-dimensional domain and the complex folded geometry of a cortical surface. There also exist several key analytical results that make a sphere an appropriate domain for validating numerical solutions against theoretical expectations. Following this, we look at how introducing a single constant delay into the standard one-dimensional neural field model affects the dynamics and stability of the solutions, before moving on to exploring several examples that incorporate the full space-dependent delay structure we wish to utilise in our later chapters. Comparing our results to those of existing authors allows us to validate that the solvers in the NFESOLVE library perform as expected.

Chapter 6 moves on to the world of neural mass models, with a focus on exploring the relationship between structural and functional connectivity in models of neural activity. Here, we aim to use an eigenmode fitting approach, based on work by Tewarie et al. [164] and Forrester et al. [65], to determine the eigenmodes of a real structural connectivity data set that best reflect the corresponding functional connectivity patterns associated with that data set. We attempt to recreate these functional connectivity patterns by generating solutions to a selection of different neural mass models, with the stability of the solutions informed by the eigenmode fitting process. We make use of the techniques seen in Chapter 3 to determine the stability of the network and locate parameter regimes which push specific eigenmodes unstable, and we employ the NFESOLVE library to generate numerical solutions to each model. After considering the traditional Wilson-Cowan model [180, 181], we end the chapter by introducing and exploring a

next-generation neural mass model, developed by Coombes and Byrne [36].

Finally, Chapter 7 explores the solution of neural field models posed on a cortical domain, with the incorporation of space-dependent delays based on real path length data. We utilise the full capabilities of the NFESOLVE library to efficiently solve a variety of models, starting with the standard neural field equation, and present a selection of example solutions on the cortex. After considering the standard neural field model, we look at the solutions yielded by introducing linear adaptation into the model. Here, we are able to visualise some interesting dynamic patterns which arise as the result of a Hopf bifurcation. To close this chapter, we build up to exploring the more intricate next-generation neural field model, adapted from the next-generation neural mass model seen in Chapter 6. This work showcases what is possible in the world of neural modelling today, displaying a variety of interesting and complex patterns on a realistic cortical domain.

Chapter 8 ends this thesis by recapitulating the work covered in each chapter, before briefly discussing the directions in which the work could be taken in the future to yield even more interesting and exciting results.

2

BACKGROUND

2.1 NEURONS, SYNAPSES AND ACTION POTENTIALS

Neurons are the fundamental processing units in the brain, receiving sensory information from the external world and relaying signals throughout the body. It is estimated that the human brain contains around 100 billion neurons [81]. Each neuron (as illustrated in Figure 1) is made up of a cell body (or soma), an axon, and dendrites. Axons are branching structures that transmit signals to other neurons. The dendrites around the cell body receive the signals coming in from the axons of other neurons via junctions called synapses. There are two types of synapses: chemical and electrical. Chemical synapses are the workhorses of neuronal information transfer [166]. They consist of a presynaptic extension to an axon called the axon terminal and a postsynaptic receiving site in the dendrites. When a signal is transmitted and the action potential (a wave of electrochemical excitation) reaches the axon terminal, this triggers the release of neurotransmitters. These are received by the neurotransmitter receptors in the dendrites, where an electrical response is initiated. A cell's mem-

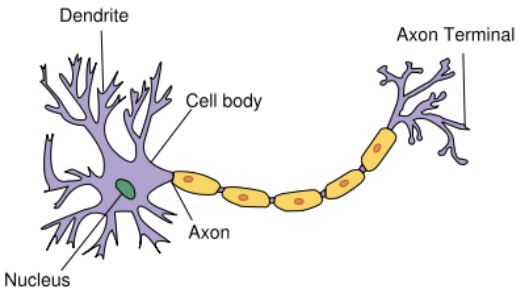


Figure 1: Illustration of a neuron showing dendrites, cell body (soma), and axon.

brane potential is defined as the difference between the electric potential inside the cell and the electric potential outside the cell. The postsynaptic response may be excitatory (increases membrane potential) or inhibitory (decreases membrane potential) depending on the type of neurotransmitters. This change in membrane potential in the receiving cell can then trigger the generation of an action potential, this is referred to as a 'spike' [98]. Unlike chemical synapses, electrical synapses form a direct physical connection between the presynaptic and postsynaptic neurons. This connection is facilitated by channels known as 'gap junctions'.

Electrical synapses are less common than chemical synapses and are somewhat simpler in their nature. They allow the flow of a current between cells via the gap junctions. These synapses transmit signals much faster than chemical synapses and allow for synchronised activity in populations of neurons due to the rapid transfer of signals from cell to cell. However, they are limited in the effects that they can have on the postsynaptic neuron, i.e., an excitatory (inhibitory) signal in the presynaptic neuron cannot induce an inhibitory (excitatory) signal in the postsynpatic neuron [160]. If the axon of a neuron synapses onto a dendrite of the same neuron, then the connection is referred to as an autapse.

2.2 INDIVIDUAL NEURON MODELLING

Although there is a vast amount about the brain that is still not yet understood, there have been considerable efforts made to comprehend and recreate observed phenomena. The mathematical modelling of brain dynamics has been a key area of interest since the mid 1900s. There have been a wide variety of models developed by numerous authors [58] and it is still a widely researched area to this day. We begin this section by exploring several existing models of individual neuron behaviour, starting with what is considered one of the most prolific and revolutionary models of its kind: the Hodgkin-Huxley model [84].

2.2.1 *Hodgkin-Huxley*

In 1952, Alan Hodgkin and Andrew Huxley studied the initiation and propagation of action potentials in the giant axon of the Atlantic squid [86, 85, 87, 84]. Through this work, they formulated what is commonly referred to today as the *Hodgkin-Huxley* model, which describes the behaviour of action potentials generated by neurons, via the flow of ions in and out of the cell. This was a cornerstone for the development of more detailed biophysical neuronal models and led to them receiving the Nobel Prize in Physiology or Medicine in 1963.

The model begins by describing the dynamics of the cell membrane potential, denoted V, by the ordinary differential equation

$$I = C\frac{\mathrm{d}V}{\mathrm{d}t} + I_i. \tag{1}$$

Here, I is the total membrane current, C is the membrane capacitance, and I_i is the current induced by the movement of ions across the membrane. Hodgkin and Huxley considered these ionic currents principally as the flow of potassium ions (K^+) and sodium ions (Na^+). The effects of all other ionic currents are considered as part of a general leak current, denoted I_l. Therefore, they write I_i as the sum

$$I_i = I_{\mathrm{K}} + I_{\mathrm{Na}} + I_l, \tag{2}$$

where I_{K} and I_{Na} are the ionic currents for the potassium and sodium ions, respectively. Each of these currents follow a simple Ohmic model, such that they can be written in terms of conductance (inverse of resistance) and voltage, i.e.,

$$I_{\mathrm{K}} = g_{\mathrm{K}}(V - V_{\mathrm{K}}), \tag{3}$$

$$I_{\mathrm{Na}} = g_{\mathrm{Na}}(V - V_{\mathrm{Na}}), \tag{4}$$

$$I_l = g_l(V - V_l), \tag{5}$$

where g_a is the conductance of the ion and V_a is the reversal potential of the ion, for $a \in \{\mathrm{K}, \mathrm{Na}, l\}$. The reversal potential for each ion is defined as the membrane potential at which there is no net flow of that ion across the membrane. These ionic conductances were then further modelled by Hodgkin and Huxley. They postulated that as well as being time-dependent, the conductances were also dependent on the voltage, V. Fitting to experimental data, they found that the conductance for the potassium ions could be modelled as

$$g_{\mathrm{K}} = \overline{g}_{\mathrm{K}} n^4, \tag{6}$$

where $\overline{g}_{\mathrm{K}}$ is the maximum value of the conductance, and n is some dimensionless quantity that varies between 0 and 1, acting as a gating variable. A gating variable is a quantity that represents the probability of an ion channel being open at any given time. The dynamics of n are given by

$$\frac{\mathrm{d}n}{\mathrm{d}t} = \alpha_n(V)(1-n) - \beta_n(V)\, n, \tag{7}$$

where $\alpha_n(V)$ and $\beta_n(V)$ are rate constants depending on the voltage, V, but not on time. They used the experimental data to determine the following expressions for these rate constants:

$$\alpha_n(V) = \frac{0.01(V+10)}{\exp\left(\frac{V+10}{10}\right) - 1}, \tag{8}$$

$$\beta_n(V) = 0.125 \exp\left(\frac{V}{80}\right). \tag{9}$$

In a similar fashion, they constructed an equation for the conductance of the sodium ions as

$$g_{\mathrm{Na}} = \overline{g}_{\mathrm{Na}} m^3 h, \tag{10}$$

where, similarly to n, the variables m and h both vary in $[0,1]$ and can be thought of as activating and inactivating gating variable, respectively. The quantity $\overline{g}_{\mathrm{Na}}$ is the maximum value of the conductance of the sodium ions. The equations for m and h take the same form as the equation for n, i.e.,

$$\frac{\mathrm{d}m}{\mathrm{d}t} = \alpha_m(V)(1-m) - \beta_m(V)\, m, \tag{11}$$

$$\frac{\mathrm{d}h}{\mathrm{d}t} = \alpha_h(V)(1-h) - \beta_h(V)\, h, \tag{12}$$

where, once again, $\alpha_{m,h}(V)$ and $\beta_{m,h}(V)$ are rate constants. Through fitting to experimental data, they yielded expressions for these rate constants, given by

$$\alpha_m(V) = \frac{0.1(V+25)}{\exp\left(\frac{V+35}{10}\right)-1}, \qquad \alpha_h(V) = 0.07\exp\left(\frac{V}{20}\right), \tag{13}$$

$$\beta_m(V) = 4\exp\left(\frac{V}{18}\right), \qquad \beta_h(V) = \frac{1}{\exp\left(\frac{V+30}{10}\right)+1}. \tag{14}$$

The leak conductance is treated as constant, i.e., $g_l = \overline{g}_l$, and does not take into consideration any gating variables. Thus, the complete four-dimensional model can be written as

$$C\frac{\mathrm{d}V}{\mathrm{d}t} = -\overline{g}_{\mathrm{K}}n^4(V-V_{\mathrm{K}}) - \overline{g}_{\mathrm{Na}}m^3h(V-V_{\mathrm{Na}}) - \overline{g}_l(V-V_l) + I, \tag{15}$$

along with Eqs. (7), (11) and (12).

This description forms the basis of many other models of individual neuron dynamics, with authors extending it to consider other phenomena such as different ionic currents [108] and stochasticity [68], to name just a few.

2.2.2 *Morris-Lecar*

Following on from the work of Hodgkin and Huxley, in 1981 Catherine Morris and Harold Lecar formulated a model to describe the electrical dynamics of the barnacle muscle fibre [127]. Although this model wasn't originally intended to be perceived in a neural context, the electrophysiology of muscle cells has a striking resemblance to that seen in neurons, leading the neuroscience community to take interest in what is now commonly referred to as the *Morris-Lecar* model [156, 167]. The model is often considered to be a simplified version of the Hodgkin-Huxley model, as it is similar in form, but notably two-dimensional as opposed to four-dimensional. Following an analogous approach to Hodgkin and Huxley, the same equation (Eq. (1)) is

used to describe the membrane potential. The ions of interest, however, are potassium (K^+) and calcium (Ca^+) ions. The total ionic current term, I_i, is given by the sum of individual currents

$$I_i = I_K + I_{Ca} + I_l, \tag{16}$$

where, once again, I_l is a leak current that accounts for all other ionic currents. These currents are described in terms of conductance and voltage via an Ohmic relationship, with the specific ionic currents being modelled dynamically, yielding

$$I_K = \overline{g}_K N(V - V_K), \tag{17}$$

$$I_{Ca} = \overline{g}_{Ca} M(V - V_{Ca}), \tag{18}$$

$$I_l = \overline{g}_l (V - V_l), \tag{19}$$

where $\overline{g}_a$ is the ionic conductance and V_a is the reversal potential of the ion, for $a \in \{K, Ca, l\}$. The recovery variables, N and M, are analogous to the gating variables, n and m, in the Hodgkin-Huxley model. Although the model originally introduced in [127] considers both N and M as dynamic, the model that is most often referred to today as the Morris-Lecar model is actually a reduction presented later on in the paper in which they consider the Ca^+ system to be so much faster than the K^+ system that M is taken to be equal to its steady state value at all times. This leads to the full model taking the form

$$C\frac{dV}{dt} = -\overline{g}_K N(V - V_K) - \overline{g}_{Ca} M_\infty(V)(V - V_{Ca}) - \overline{g}_l (V - V_l) + I, \tag{20}$$

$$\frac{dN}{dt} = \alpha_N(V)(N_\infty(V) - N), \tag{21}$$

where the quantities M_∞, N_∞ and α_N are given the forms

$$M_\infty(V) = \frac{1}{2}\left(1 + \tanh\left(\frac{V - V_1}{V_2}\right)\right), \tag{22}$$

$$N_\infty(V) = \frac{1}{2}\left(1 + \tanh\left(\frac{V - V_3}{V_4}\right)\right), \tag{23}$$

$$\alpha_N(V) = \alpha_0 \cosh\left(\frac{V - V_3}{2V_4}\right), \tag{24}$$

as described by Lecar et al. several years earlier in [114, 53].
This model has been used extensively to describe the dynamics of fast-spiking neurons. It can yield a variety of solution behaviours, depending on parameter values, which is illustrated by the fact that the nullclines of the model are instinctively non-linear and can therefore interact in a multitude of ways. A key characteristic it possesses is the ability to generate sharp changes in membrane potential if a specific voltage threshold is reached. The negative feedback mechanism governed by Eq. (21) then works to bring the excitable neuron back down to resting-state potential. Although this model is simpler than the Hodgkin-Huxley model, it can still yield solutions that are just as reflective of observed neuronal dynamics.

2.2.3 *QIF/Theta Neuron*

Taking a different approach from the models discussed above, the quadratic integrate-and-fire (QIF) model is a purely phenomenological model which naturally yields solutions of a spiking nature. It is constructed as

$$\frac{\mathrm{d}V}{\mathrm{d}t} = V^2 + I, \tag{25}$$

where V represents voltage and I represents some external current. With I a positive constant, solutions to this differential equation all take the form of the tangent function. Asymptotically, when the solution reaches positive infinity it undergoes a reset back to negative infinity, thus yielding a spike. Numerically, we say that when the solution reaches a threshold value, V_{th}, at a given time, T, it undergoes a reset back down to a value, V_{R}. Denoting each spike time with a subscript index, $i = 0, 1, \ldots$, this reset rule can be written into the model as

$$\frac{\mathrm{d}}{\mathrm{d}t} V(t) = V^2(t) + I, \tag{26}$$

$$V\left(T_i^-\right) = V_{\mathrm{th}}, \tag{27}$$

$$V\left(T_i^+\right) = V_{\mathrm{R}}, \tag{28}$$

where $T_i^{\pm} = \lim_{\epsilon \to 0} (T_i \pm \epsilon)$.

In 1986, Ermentrout and Kopell [57] published the Ermentrout-Kopell canonical model, or theta neuron model as it is otherwise known. This model is posed on a unit circle and is used to simulate the spikes of a neuron. Consider a point on the circle with angular coordinate θ, for $\theta \in [0, 2\pi)$ radians. The model is treated in such a way that it is said that whenever $\theta = \pi$ an action potential (or spike) is generated. It takes the form

$$\frac{\mathrm{d}\theta}{\mathrm{d}t} = 1 - \cos(\theta) + I(1 + \cos(\theta)), \tag{29}$$

where I is an external input. It can be shown that the QIF model, under the change of variables $V = \tan(\theta/2)$, is mathematically equivalent to the theta neuron model

This model has since been studied in a variety of settings. One such study is by Ermentrout et al. [55], where they explore the emergence of travelling wave solutions in a network of theta neurons, with a focus on the conditions of existence for such classes of solutions. More recently, it has been employed by Coombes and Byrne [36] as a basis for the development of their next-generation neural mass model, which we explore in detail in Chapter 6. It even been used in such applications as artificial intelligence, where it has been used to develop a multilayer neural network with a learning rule based on intrinsic neural dynamics [122].

2.3 MODELLING AT THE MACROSCOPIC LEVEL

The models discussed thus far have only considered the dynamics of single neurons. Although this work is useful for describing neural dynamics at the microscopic level, it becomes implausible to consider the dynamics of each individual neuron when looking at how the brain behaves at the macroscopic level, due to the brain containing such a large number of neurons. In

this section, we introduce and discuss several models which aim to describe how populations of neurons behave across the cortex, thereby constructed as being dependent not only on time, but having a spatial component too. The neural field models presented here are pertinent to the rest of the work in this thesis, and the principles discussed here are used to construct the next-generation neural field model that we explore in Chapter 7.

2.3.1 *Wilson-Cowan Model*

One of the most well known models of neuronal activity for populations of neurons is the Wilson-Cowan model, developed by Hugh Wilson and Jack Cowan in 1972 [180, 181]. They considered the interaction between a population of excitatory neurons, E, and a population of inhibitory neurons, I, coupled together. Excitatory (inhibitory) neurons increase (decrease) the likelihood that their neighbouring neurons will become active, with the activation being governed by a nonlinear function, $F_{E,I}$. This is typically taken to be sigmoidal in shape. Although the model was originally conceived for a network of neurons, the continuum form of the model is given by

$$\tau_E \frac{\partial}{\partial t} E(x,t) = -E(x,t) + (1 - rE(x,t)) F_E(w_{EE} \otimes E - w_{EI} \otimes I + Q_E(x,t)), \tag{30}$$

$$\tau_I \frac{\partial}{\partial t} I(x,t) = -I(x,t) + (1 - rI(x,t)) F_I(w_{IE} \otimes E - w_{II} \otimes I + Q_I(x,t)). \tag{31}$$

Here w_{ab} is a connectivity kernel representing connections from population a to population b, where $a, b \in \{E, I\}$, and $\otimes$ represents the spatial convolution

$$[w_{ab} \otimes A](x,t) = \int_\Omega w_{ab}(|x - x'|) A(x',t) \, \mathrm{d}x'. \tag{32}$$

This convolution structure is built on the assumption that interactions only depend on distance. The terms $Q_{E,I}$ are external inputs, and $\tau_{E,I}$ are temporal scaling parameters. The system also incorporates the refractory dy-

namics of both populations with the terms $(1 - rA(x,t))$, $A \in \{E, I\}$. This, however, is often omitted from newer considerations of the model.

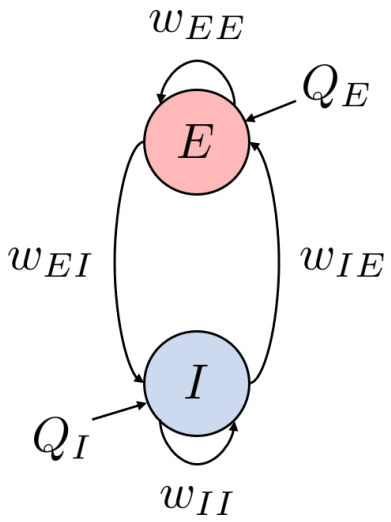


Figure 2: Schematic of the Wilson-Cowan model showing the excitatory and inhibitory populations and the inputs they receive.

We explore this model later in Chapter 6, building upon work by Tewarie et al. [164] and Forrester et al. [65, 66], who use the model to explore the relationship between structural and functional connectivity in the brain (see Section 2.4 for an introduction to these connectivity measures). For more details on the construction and applications of the Wilson-Cowan model see [35].

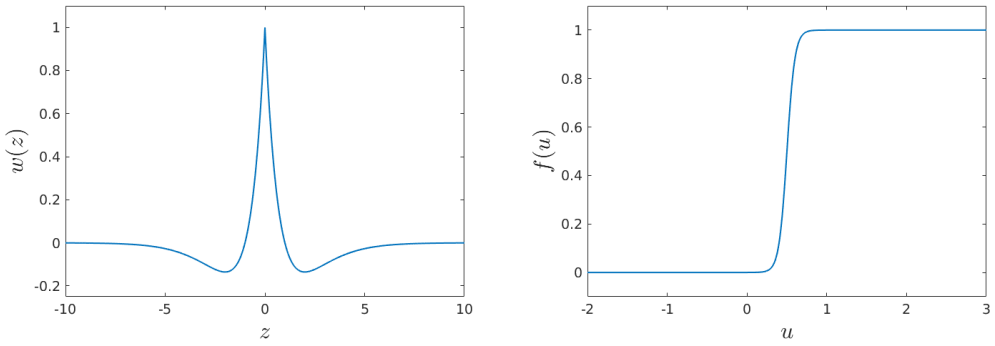
2.3.2 *Neural Field Models*

Neural field equations (NFEs) model the spatiotemporal evolution of synaptic or firing rate activity in populations of neurons at the tissue level. In 1977, Amari [2] formulated a single population voltage-based model for activity, $u = u(\boldsymbol{x}, t)$, posed on a continuous domain, $\Omega \subseteq \mathbb{R}^d$. The domain, $\Omega \subseteq \mathbb{R}^d$,

is typically considered only for $d = 1, 2, 3$ as it would not make physical sense to consider a higher dimensional domain. The model takes the form

$$\frac{\partial}{\partial t} u(\boldsymbol{x}, t) = -u(\boldsymbol{x}, t) + \underbrace{\int_{\Omega} w(\boldsymbol{x}, \boldsymbol{x}') f\left(u\left(\boldsymbol{x}', t - \tau(\boldsymbol{x}, \boldsymbol{x}')\right)\right) \mathrm{d}\boldsymbol{x}' + I(\boldsymbol{x}, t)}_{\varrho}. \tag{33}$$

We shall briefly explain the formulation of this model. With no input ($\varrho = 0$), activity decays exponentially. The kernel, w, represents the connectivity of points on the domain, Ω, such that for any two points, $\boldsymbol{x}$ and $\boldsymbol{x}'$, the strength of the connection between them is $w(\boldsymbol{x}, \boldsymbol{x}')$. For simplicity, this is often chosen to be a function of distance between the two points, i.e., $w(\boldsymbol{x}, \boldsymbol{x}') = w(\|\boldsymbol{x} - \boldsymbol{x}'\|)$. From a neuroscience perspective, this kernel can be a connectome (a comprehensive map of the macroscopic connections as estimated by diffusion MRI [103]). If the values are positive then they represent excitatory connections and if they are negative they represent inhibitory connections. Typically, kernels which have short range excitatory connections and long range inhibitory connections are referred to as 'Mexican hat' or 'wizard hat' connectivity kernels [35] (see Figure 3a). The firing rate (or activation function) is represented by f, which is usually chosen to have a sigmoidal shape [180, 38] (see Figure 3b) and a range of $[0, 1]$. For steep sigmoids, this acts as a switch that "turns on" the contribution of points in the network when the activity at those points crosses a threshold. The connectivity kernel and firing rate are multiplied together and then summated across the whole domain (i.e., we integrate over $\boldsymbol{x}'$) to give the basic neural field model. The term, $I(\boldsymbol{x}, t)$, represents an external stimulus.



(a) Connectivity kernel $w(z)$.

(b) Firing rate $f(u)$.

Figure 3: Examples of a wizard hat connectivity kernel, $w(z) = (1 - |z|)\,\mathrm{e}^{-|z|}$, and sigmoidal firing rate given by Eq. (34), with $\mu = 20$, $\theta = 0.5$.

A common choice of firing rate [35] is

$$f(u) = \frac{1}{1 + \mathrm{e}^{-\mu(u-\theta)}}, \tag{34}$$

which we will use throughout this work. The parameter, μ, controls the steepness of the sigmoid and the parameter, θ, controls the threshold at which the neurons become active. In order to make analytical progress, sigmoidal firing rates with large μ are approximated by the Heaviside step function defined by

$$H(u-\theta) = \begin{cases} 1, & \text{if } u \geq \theta, \\ 0, & \text{if } u < \theta, \end{cases} \tag{35}$$

which is obtained from Eq. (34) in the limit as $\mu \to \infty$. Amari's work [2] focussed heavily on the dynamical behaviour of the system with the use of a Heaviside activation function, showing the existence and stability of a class of solutions known as 'bump' solutions, as well as travelling bump solutions. These types of solutions, as the name suggests, take the form of spatially localised bumps, similar in appearance to Mexican hat kernels.

Although the original development of the Amari model did not include axonal delays, they have become an increasingly studied addition to

the model. Axonal delays are prevalent in the brain due to the propagation of signals not being instantaneous. When an action potential is generated, it travels along the axon of the neuron it is generated from to reach the dendrites of another neuron. The velocity of action potentials along axons can vary from $1\mathrm{m\,s^{-1}}$ to over $100\mathrm{m\,s^{-1}}$ [51], although typically velocities average at around $10\mathrm{m\,s^{-1}}$ in the cortex. For simplification, many models assume that these propagation speeds are infinite. However, this is not as physically representative. Axonal delays in NFEs have been studied at length by a number of authors, such as in [105, 91, 92, 34, 61, 142, 111, 90]. The physical relevance of delays means that they are commonly selected such that $\tau(\boldsymbol{x}, \boldsymbol{x}') = \|x - x'\|/v$, hence making them spatially dependent. Here, $\|\boldsymbol{x} - \boldsymbol{x}'\|$ is some measure of distance between two spatial points, and v is a propagation speed. Physically, this construction represents the observed transmission delay of signals propagating along axons, with the time taken being directly proportional to the length of the axon. Although v is assumed constant here, in practice the degree of myelination on different axonal fibers could affect the propagation speed. The myelin sheath is a fatty substance (yellow blobs in Figure 1) that insulates axons to help increase the speed at which signals travel. Atay and Hutt [6] have studied the incorporation of statistically distributed propagation speeds into models of large-scale brain activity, along with non-local delayed feedback loops. Some studies (e.g. [174, 164]) also include a constant delay term that applies equally to all signals. This results in the delay expression taking the form $\tau(\boldsymbol{x}, \boldsymbol{x}') = \tau_c + \|x - x'\|/v$, where τ_c is the fixed delay. The role that fixed delays play in neural modelling is discussed in detail by Montbrio and Roxin in [125]. They attribute it to a synaptic/dendritic processing delay; however, as evident from the wide range of studies listed above that do not include this constant delay term, it is not clear whether this is actually physically relevant. A non-physical reason why it can be incorporated into firing rate models is that it can allow for dynamics similar to those seen in simulations of large-scale spiking networks [142]. Throughout this thesis

we include the constant delay term in our presentations of delay models, on the understanding that for a purely physical representation this would ideally be set to zero.

Incidentally, delays also allow for significantly more interesting dynamics to be prevalent in solutions, bringing them closer to the observed naturally occurring brain dynamics that we wish to model. It was shown by Nunez in 1974 [129] that for certain choices of the delays, τ, and connectivity kernel, w, the model can be reduced to a form of the wave equation, known as the *brain wave equation*. Therefore, travelling wave solutions can also occur more naturally with the presence of axonal delays.

2.3.3 *Neural Field With Linear Adaptation*

The Amari equation Eq. (33) can be extended to incorporate an intrinsic negative feedback mechanism, known as spike-frequency adaptation, that neural populations possess which brings them back to rest levels after periods of high activity [135]. This model is based on work by Hansel and Sompolinsky [79], where they consider a network in which the excitatory cells are endowed with spike-frequency adaptation. This was then further extended by several authors such as [42, 110, 135, 37] to yield the following model.

$$\tau \frac{\partial}{\partial t} u(\boldsymbol{x}, t) = -u(\boldsymbol{x}, t) - g a(\boldsymbol{x}, t) + \int_{\Omega} w(\boldsymbol{x}, \boldsymbol{x}') f\left(u(\boldsymbol{x}', t)\right) \mathrm{d}\boldsymbol{x}', \tag{36}$$

$$\frac{1}{\alpha} \frac{\partial}{\partial t} a(\boldsymbol{x}, t) = u(\boldsymbol{x}, t) - a(\boldsymbol{x}, t). \tag{37}$$

The parameter, g, is the coupling strength of the feedback, and α is the rate at which adaptation occurs. If g is set to 0 then the system is equivalent to the standard Amari neural field equation Eq. (33). The parameters, τ and α^{-1}, are timescales. In [56], Ermentrout, Folias, and Kilpatrick study the spatiotemporal pattern formations for this model, providing analyses for both periodic and infinite domains in one and two dimensions. They showed that there exists multiple dynamic solution patterns, such as travel-

ling waves, 'breathers' and 'sloshers'. Breather solutions consist of spatially localised oscillations such that a bump solution appears to expand and contract, while slosher solutions are bumps that move or 'slosh' from side to side. Other pattern formation and bifurcation analyses have also been undertaken in such works as [39], where the authors develop a linear stability analysis for the interface dynamics between areas of high and low activity in a planar neural field model; and [64], in which the dynamics of 'breather' solutions in an excitatory neural network are explored.

2.3.4 *PDE Formulations of Neural Fields*

Although neural fields are typically studied in the integro-differential form Eq. (33), for suitable choices of connectivity kernel w it is possible to transform the model into a partial differential equation (PDE); see, for example, [113]. The non-delay neural field in one spatial dimension can be written as

$$\left(\frac{\partial}{\partial t}+1\right) u\left(x,t\right)=\int_{-\infty}^{\infty} w\left(x-y\right) f\left(u\left(y,t\right)\right) \mathrm{d}y. \tag{38}$$

The Fourier transform is defined for a function $u=u\left(x,t\right)$ as

$$F[u\left(x,t\right)]\left(k,t\right)=\int_{-\infty}^{\infty} \mathrm{e}^{ikx} u\left(x,t\right) \mathrm{d}x, \tag{39}$$

where k is the transform variable. Using the convolution theorem, the Fourier transform can be applied to both sides of Eq. (38) to remove the spatial component. This gives

$$\left(\frac{\partial}{\partial t}+1\right) F\left[u\right]\left(k,t\right)=F\left[w\right]\left(k\right)\times F\left[f\left(u\right)\right]\left(k,t\right), \tag{40}$$

where $\times$ represents standard multiplication. Suppose that the Fourier transform of w is a rational function of k^2, i.e.

$$F\left[w\right]\left(k\right)=\frac{P\left(k^2\right)}{Q\left(k^2\right)}, \tag{41}$$

where P and Q are polynomials. Multiplying Eq. (40) by $Q\left(k^2\right)$ gives

$$\left(\frac{\partial}{\partial t}+1\right) Q\left(k^2\right) F\left[u\right]\left(k,t\right)=P\left(k^2\right) F\left[f\left(u\right)\right]\left(k,t\right). \tag{42}$$

Recalling that, under the assumption of vanishing initial data, the Fourier transform of the second and fourth spatial derivatives of a function $u(x,t)$ are $-k^2F[u](k,t)$ and $k^4F[u](k,t)$ respectively, taking the inverse Fourier transform of Eq. (42) gives

$$\left(\frac{\partial}{\partial t}+1\right)D_1u(x,t)=D_2f(u(x,t)), \tag{43}$$

where D_1 and D_2 are linear differential operators in space that are of even order, associated with Q and P, respectively [112].

To give an example, say we have the wizard hat function $w(x) = (1-|x|)\,e^{-|x|}$, depicted in Figure 3. This has Fourier transform $4k^2/(1+k^2)^2$, allowing us to write the NFE as a PDE of the form

$$\left(1+\frac{\partial}{\partial t}\right)\left(1-\frac{\partial^2}{\partial x^2}\right)^2u(x,t)=-4\frac{\partial^2}{\partial x^2}f(u(x,t)). \tag{44}$$

This technique can also be applied to the full delay problem, as seen in [104] where the equation was first derived. Coombes et al. study the PDE formulation of the full space-dependent delay problem for a two dimensional domain in [40]. Doing so, they obtain a damped inhomogeneous wave equation.

When solving NFEs numerically, the PDE formulation can be less computationally challenging to work with than the integral formulation, making it appealing. Typically a finite difference scheme [94] would be applied to the spatial derivatives. When the equations are then fully discretised the resulting system involves fairly sparse matrices as opposed to the dense matrices that the connectivity kernel in the integral formulation generates. This allows for optimised matrix-vector operations that remove unnecessary computations due to the large number of zeros the matrices contain.

2.4 MEASURING MACROSCOPIC BRAIN CONNECTIONS

One of the objectives of this thesis is to develop, simulate, and analyse neural field models posed on a two dimensional cortical domain embedded

in $\mathbb{R}^3$. A fundamental aspect of these models is the choice of a connectivity kernel. There is a vast amount of work already done in the analysis of these problems where the kernel is chosen to be a specified function [33, 63, 64, 140]. A more realistic approach that we are taking in this project is to utilise proxies of brain connectivity available from Magnetic Resonance Imaging (MRI). MRI data collected on real human subjects can provide estimates of brain connections in a number of ways [103, 159] and little development with incorporating such information into neural field models has been done thus far. In particular, the Human Connectome Project [172, 173] (`https://www.humanconnectome.org`) has spent many years studying and gathering data from a range of subjects, with the aim of effectively mapping macroscopic brain connections and their variability across healthy adults [158] and making all the data publicly available. With advances in MRI technology, the acquisition of imaging data in a non-invasive way has become a fundamental component for providing insight into brain function and the changes that occur with development, aging, and disease [116].

2.4.1 *Diffusion MRI*

An important technique used to measure structural connectivity patterns in the brain is diffusion MRI (dMRI), as opposed to functional MRI (fMRI) which is used for measuring brain activity. Although structure can be inferred from function by addressing statistical dependencies in the activity between regions, the anatomical routes the physical axonal connections take remain unknown. Diffusion MRI works by detecting the random thermal motion of water molecules [103, 159]. In organised structure, such as white matter, the diffision of water molecules is anisotropic. This means that it they are less hindered when moving parallel to axonal fibres than when moving perpendicular to them. Measuring the orientational dependence of the water diffusion allows for the estimation of the axonal orientations, which are then used by post-processing algorithms known as *tractography*

[158] to infer long range connections. Tractography works by tracking the flow of the diffusion through areas of the white matter domain, known as voxels, until a boundary or specified termination point is reached. Repeating the tractography for many locations in the brain allows for the construction of a structural connectome (a comprehensive map of the physical connections between regions of the brain). Although dMRI cannot directly quantify strength of connections, it does allow for estimation of edge weights that can reflect desired properties, such as axonal density, myelination, etc. A common edge weighting quantification is to look at the size of axonal fibre bundles inferred from the tractography [159]. This data can then be represented in the form of a structural connectivity matrix, an example of which is shown in Figure 4. Along each axis are the locations for which the tractography is repeated at. The coloured pixels denote the strength of the connections between each location (red is strongly connected, blue weakly connected). Note the symmetry about the main diagonal of this matrix; this is due to the nature of measuring diffusion MRI, as the signal is antipodally symmetric (diffusion along x and $-x$ induces the same MR signal) and therefore estimated connections are not directional.

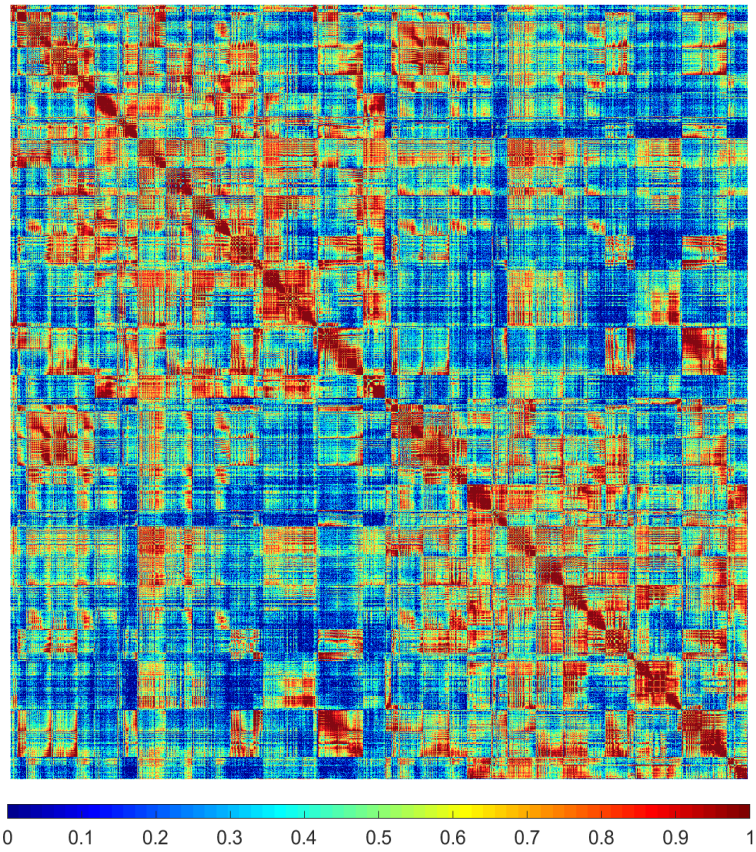


Figure 4: Example of a dense connectome matrix computed using probabilistic tractography on diffusion MRI data from 50 Human Connectome Project subjects [72, 173]. The colour represents the magnitude of the edge weights assigned to each connection. Areas coloured in red represent strong structural connections and areas coloured in blue depict weak or non-existent structural connections. The connection strengths have been normalised to the range $[0, 1]$.

It is also largely unknown as to when erroneous connections in the tractography occur. As there can be hundreds of thousands of axons, the mapping from diffusion to axonal directions is often ill-posed. This results in the tractography algorithms producing false positive and negative connections due to taking 'wrong turns'. We refer to [171] and [159] for discussions on the challenges faced with employing dMRI.

2.4.2 *Functional MRI*

Functional MRI (fMRI) is a process used frequently by both physicians and researchers to gather data on brain activity patterns. Although similar to standard MRI, the major difference is that fMRI is specifically used to measure the changes in blood flow in the brain. When neuronal activity in specific areas starts to increase, there is a higher demand for oxygen in those regions, thereby requiring an increase in blood flow to deliver it. Oxygen is carried in the red blood cells by the protein named haemoglobin. When it reaches the required destination, the oxygenated haemoglobin replaces the deoxygenated haeomglobin. It is the difference in magnetic properties between these molecules that allows for blood flow to be measured using fMRI. The oxygenated haemoglobin is diamagnetic, meaning it is far less sensitive to a magnetic field than the deoxygenated haemoglobin, which is paramagnetic. This contrast is known as the blood-oxygen-level dependent (BOLD) signal [74, 169]. As the neurons fire, the changes in blood flow are detected by the MRI machine, leading to a visualisation of the areas of the brain that are concurrently active, thereby indicating a functional connection between those regions. Additionally, the temporal correlation between these signals may be analysed via correlations measures, such as Pearson's correlation coefficient [134] or the Jaccard similarity [99], to construct a functional connectome [67, 75]. This is often represented in a similar fashion to structural connectivity data, in the form of a matrix.

2.5 NUMERICAL INTEGRATION METHODS

As introduced above in Section 2.3, macroscopic models of neural activity typically contain an integral term to facilitate the summation of neural connections across the cortex as a continuum. This poses a challenge when solving these models numerically, and as such, specialist numerical meth-

ods are required to numerically approximate these integral terms. Although we shall illustrate later in Section 3.3 how a neural field may be fully discretised and numerically solved, this section gives an introduction to the numerical integration methods which will be utilised.

2.5.1 *Quadrature*

A quadrature rule is a form of approximation for a definite integral of a function. Quadrature rules work by taking a weighted sum of function points evaluated at a specific set of abscissas (points) across the domain of integration [54]. Conventionally, the domain of integration for a rule, in one dimension, is specified as the interval $[-1, 1]$. For a function, $g(x)$, a quadrature rule to approximate the integral of that function takes the form

$$\int_{-1}^{1} g(x)dx \approx \sum_{i=1}^{N} w_i g(x_i). \tag{45}$$

In order to numerically compute integrals on a general domain, a mapping can be employed to map from the general domain onto the reference domain.

2.5.2 *Gauss-Legendre Quadrature*

There are many quadrature rules which require different classes of orthogonal polynomials, such as Gauss-Jacobi quadrature, Chebyshev-Gauss quadrature and Newton-Cotes quadrature [138]. One of the most widely used quadrature rules is Gauss-Legendre quadrature. Developed by Gauss [70] in the early 19th century, it was then noted by Jacobi [101, 102] twelve years later that the quadrature points are the roots of the Legendre polynomials [115].

The Legendre polynomials are a set of functions that are orthogonal on the interval $(-1, 1)$. They have several different definitions resulting in

the same equations; however, one way of generating them is by using the following recursive formula:

$$\begin{aligned} L_0(x) &= 1, \\ L_1(x) &= x, \\ L_{n+1}(x) &= \frac{2n+1}{n+1} x L_n(x) - \frac{n}{n+1} L_{n-1}(x), \quad n \geq 1. \end{aligned} \tag{46}$$

The quadrature points (abscissas), x_i, $i = 1, \ldots, n$, for a Gauss-Legendre quadrature rule of order n, are given by the roots of the nth order Legendre polynomial $L_n(x)$, i.e., the solution to $L_n(x) = 0$. To calculate these roots numerically using a root-finding algorithm, the Chebyshev nodes [32], given by the formula

$$x_i = -\cos\left(\frac{2i-1}{2n}\pi\right), \tag{47}$$

where $i = 1, \ldots, n$, can be used as an initial guess as they sit close to the Legendre nodes.

The weights, w_i, $i = 1, \ldots, n$, corresponding to the points of the quadrature rule, are defined by

$$w_i = \frac{2}{(1-x_i^2)[L_n'(x_i)]^2}, \tag{48}$$

where $L_n'(x)$ is the derivative of the nth order Legendre polynomial. These derivatives can also be generated using a recursive formula:

$$\begin{aligned} L_0'(x) &= 0, \\ L_1'(x) &= 1, \\ L_{n+1}'(x) &= \frac{2n+1}{n} x L_n'(x) - \frac{n+1}{n} L_{n-1}'(x), \quad n \geq 1. \end{aligned} \tag{49}$$

Using these points and weights in Eq. (45), this rule can yield an exact solution to definite integrals for which the integrand is a polynomial of degree $2n-1$.

This introduction to Gauss-Legendre quadrature is for an interval domain; however, a detailed explanation of the formulation of the Gauss-Legendre quadrature rule on polygons in higher dimensions, such as triangles and quadrilaterals, can be found in [49].

3

ANALYTICAL AND NUMERICAL TECHNIQUES FOR NEURAL MODELLING

3.1 INTRODUCTION

There are many methodologies that researchers employ when working with mathematical models of real-world phenomena. These range from analytical techniques, such as linear stability analysis, that are used to gain an insight into how a system may behave under certain conditions, to the numerical schemes implemented to approximate solutions computationally. Before exploring specific models of neural activity, it is important to consider the key analytical and numerical approaches used in neural modelling.

To start, we introduce an arbitrary network of nodes whose dynamics are governed by generic non-linear functions. Many neural mass models follow this structure, as we will see later in Chapter 6. We carry out a linear stability analysis of the general model and show how the linearised equations can be broken down with respect to the eigenvalues of the global coupling matrix present in the system. Following on from this, we discuss how the presence of delays impacts on the system and its stability.
The next section of this chapter then introduces the numerical techniques required to solve continuum models of neural activity. These models typically contain integral terms to facilitate long-range connections, thereby

not only necessitating a spatial discretisation of the continuum, but also requiring a quadrature rule to numerically approximate the integral. We study a general quadrature rule, applicable to all spatial domains, along with defining a specific Gauss quadrature rule on interval domains. We then illustrate its use by spatially discretising a standard NFE and turning it into a system of purely time-dependent coupled equations. Finally, a convergence analysis of the discussed methods is presented, showing how the NFE performs under different discretisations and quadrature accuracies.

3.2 ANALYSIS OF ARBITRARY NON-LINEAR NETWORKS

In Chapter 6, we consider a selection of neural mass models used to describe the dynamics of large-scale brain networks. A desired component of this work is to generate model solutions of an oscillatory nature. In order to facilitate locating these solutions, we present here the construction of an arbitrary network of nodes in a generalised form, and use linear stability analysis techniques to derive the system's spectral equation of eigenvalues in terms of the decomposed non-local connectivity matrix that governs the connections between the nodes in the network. This linear stability analysis is important as it is utilised to locate parameter regimes for which a chosen system undergoes a Hopf bifurcation, thereby resulting in the emergence of oscillatory dynamics. Although the techniques presented here are included primarily for the application detailed in Chapter 6, the general nature of the construction allows this analysis to be applied to other applications in neural modelling where linear stability analysis is required. This section is based on the supplementary material of [164].

3.2.1 *Construction of an Arbitrary Non-Linear Network*

Consider a set of neural populations $x(t) = \left(x^1(t), \ldots, x^m(t)\right) \in \mathbb{R}^m, t \geq 0$, whose dynamics are governed locally by the equation

$$\frac{\mathrm{d}}{\mathrm{d}t}x = F(x) + G\left(w^{\mathrm{loc}}x\right), \tag{50}$$

where $F, G : \mathbb{R}^m \mapsto \mathbb{R}^m$, and $w^{\mathrm{loc}x} \in \mathbb{R}^{m\times m}$ determines the strength of the local interactions between populations. Using this, a network of N interconnected nodes each comprising m populations can be constructed, with equations

$$\frac{\mathrm{d}}{\mathrm{d}t}x_i = F(x_i) + G\left(w^{\mathrm{loc}}x_i + s_i\right), \quad s_i = \sum_{j=1}^{N} w_{ij}H\left(x_j\right), \tag{51}$$

where $x_i = \left(x_i^1, \ldots, x_i^m\right)$, $i = 1, \ldots, N$, w_{ij} are the entries of a connectivity matrix that describes the strength of connections between individual nodes, and $H : \mathbb{R}^m \mapsto \mathbb{R}^m$ is a linear mapping that governs which populations have a cross-nodal effect on entire network.

3.2.2 *Linear Stability Analysis*

The steady state of the network is given by

$$0 = F(\overline{x}_i) + G\left(w^{\mathrm{loc}}\overline{x}_i + \overline{s}_i\right), \quad \overline{s}_i = \sum_{j=1}^{N} w_{ij}H\left(\overline{x}_j\right), \tag{52}$$

where $\overline{x}_i = \left(\overline{x}_i^1, \ldots, \overline{x}_i^m\right)$, which represent the steady states of each population at node i. Assuming a small perturbation of the form $x_i(t) = \overline{x}_i + u_i\,\mathrm{e}^{\lambda t}$, where $u_i \in \mathbb{R}^m$, and $\lambda \in \mathbb{C}$, the steady state can be linearised around to give

$$\lambda u_i = \left[DF(\overline{x}_i) + DG\left(w^{\mathrm{loc}}\overline{x}_i + \overline{s}_i\right)w^{\mathrm{loc}}\right]u_i + \sum_{j=1}^{N} DG\left(w^{\mathrm{loc}}\overline{x}_i + \overline{s}_i\right)DH\left(\overline{x}_j\right)w_{ij}u_j, \tag{53}$$

where $DF, DG, DH \in \mathbb{R}^{m\times m}$ are Jacobian matrices. For convenience, this can be written in the shortened form

$$\lambda u_i = D\widetilde{F}_i u_i + \sum_{j=1}^{N} D\widetilde{G}_i w_{ij} u_j, \tag{54}$$

where

$$\begin{bmatrix} D\widetilde{F}_i \\ D\widetilde{G}_i \end{bmatrix} = \begin{bmatrix} DF\left(\overline{x}_i\right) + DG\left(w^{\mathrm{loc}}\overline{x}_i + \overline{s}_i\right) w^{\mathrm{loc}} \\ DG\left(w^{\mathrm{loc}}\overline{x}_i + \overline{s}_i\right) DH\left(\overline{x}_j\right) \end{bmatrix}. \tag{55}$$

It must be noted here that $DH\left(\overline{x}_j\right)$ is independent of the label j due to the fact that H is linear. Hence, the entire equation can be written succinctly as

$$\lambda U = \begin{pmatrix} D\widetilde{F}_1 & & 0 \\ & \ddots & \\ 0 & & D\widetilde{F}_N \end{pmatrix} U + \begin{pmatrix} D\widetilde{G}_1 & & 0 \\ & \ddots & \\ 0 & & D\widetilde{G}_N \end{pmatrix} (w \otimes \mathcal{I}_m)\, U, \tag{56}$$

where $U = \left(u_1^1, \ldots, u_1^m, \ldots, u_N^1, \ldots, u_N^m\right)^T$, the operator $\otimes$ is the Kronecker product, and $\mathcal{I}_m$ is the $m \times m$ identity matrix. In this form, this is as far as we are able to reasonably progress with the linear stability analysis of a generalised network without making simplifications and assumptions. Depending on the specific model under consideration, numerically computing the eigenvalues of the system at this stage may be implausible. Factors that could contribute to this are the size of the system and the heterogeneity of the steady state.

In order to take this analysis further, we make the assumption that the rows of w are normalised such that $\sum_j w_{ij} = 1$ for all $i = 1, \ldots, N$. This step allows for the guaranteed existence of a homogeneous steady state $\overline{x}_i = \overline{x}$ for all $i = 1, \ldots, N$, thereby removing the dependence on i in $D\widetilde{F}_i$ and $D\widetilde{G}_i$. Diagonalising w, it decomposes into the form $w = P\Lambda P^{-1}$, where P is the matrix of normalised eigenvectors of w and Λ is the diagonal matrix of eigenvalues $\mu_1, \ldots, \mu_N$. Employing the change of variables $V = (P \otimes \mathcal{I}_m)^{-1} U$, and manipulating the expression using the properties

of the Kronecker product (see [164] for the full derivation), the system can be written in terms of the eigenvalues of w as

$$\lambda V = \begin{pmatrix} D\widetilde{F} & & 0 \\ & \ddots & \\ 0 & & D\widetilde{F} \end{pmatrix} V + \begin{pmatrix} \mu_1 D\widetilde{G} & & 0 \\ & \ddots & \\ 0 & & \mu_N D\widetilde{G} \end{pmatrix} V. \tag{57}$$

The block diagonal form of the system means that it can be split into N independent equations of the form

$$\lambda V_p = \left[D\widetilde{F} + \mu_p D\widetilde{G} \right] V_p, \quad p = 1, \ldots, N, \tag{58}$$

where $V_p \in \mathbb{C}^m$. The eigenvalue spectrum for the whole network can be generated by finding solutions to

$$\mathcal{E}(\lambda; p) := \left| \lambda \mathcal{I}_m - D\widetilde{F} - \mu_p D\widetilde{G} \right| = 0, \quad p = 1, \ldots, N. \tag{59}$$

3.2.3 *Incorporation of Delays*

When delays are present in the network, equation Eq. (51) becomes

$$\frac{\mathrm{d}}{\mathrm{d}t} x_i = F(x_i) + G\left(w^{\mathrm{loc}} x_i + s_i \right), \quad s_i = \sum_{j=1}^{N} w_{ij} H\left(x_j\left(t - \tau_{ij} \right) \right). \tag{60}$$

Here, τ_{ij} represents the delay value between node i and node j. The steady state of the system remains the same; however, the linearised equation becomes

$$\lambda u_i = D\widetilde{F} u_i + D\widetilde{G} \sum_{j=1}^{N} \widetilde{w}_{ij}(\lambda) u_j, \tag{61}$$

where $\widetilde{w}(\lambda)_{ij} = w_{ij} \mathrm{e}^{-\lambda \tau_{ij}}$. We further assume that $\widetilde{w}_{ij}(\lambda)$ can be decomposed into the form

$$\widetilde{w}_{ij}(\lambda) = \sum_{p=1}^{N} \mu_p(\lambda) \gamma_j^p \zeta_i^p, \tag{62}$$

where γ^p and ζ^p are normalised left and right eigenvectors of w, respectively, such that they form a dual basis of the eigenspace of w. By projection, the coefficients $\mu_p(\lambda)$ can be generated as

$$\mu_p(\lambda) = \sum_{i=1}^{N} \sum_{j=1}^{N} \widetilde{w}_{ij}(\lambda) \gamma_i^p \zeta_j^p. \tag{63}$$

Under the replacement $\mu_p \to \mu_p(\lambda)$ in Eq. (59), the eigenvalue spectrum of the delayed network is given by solutions to

$$\mathcal{E}(\lambda; p) := |\lambda \mathcal{I}_m - D\widetilde{F} - \mu_p(\lambda) D\widetilde{G}| = 0, \quad p = 1, \ldots, N. \tag{64}$$

The incorporation of delays into the system turns the eigenvalue problem, $\mathcal{E}(\lambda; p)$, into a transcendental equation. To solve this, a numerical root-finding algorithm is required.

3.3 NUMERICAL DISCRETISATION OF CONTINUUM MODELS

In the continuum limit of models of neural activity (see Section 2.3), the summation of long-range connections becomes an integral. This turns the models into a system of integro-differential equations. Typically, these equations are posed on domains in $\mathbb{R}^d$, $d = 1, 2, 3$. In this thesis, we primarily consider intervals in $\mathbb{R}^1$, areas in $\mathbb{R}^2$, and surfaces in $\mathbb{R}^3$.

3.3.1 *Finite Element Mesh*

The first step in the numerical solution of continuum models is partitioning the domain into a finite element mesh. Let Ω denote the continuum domain, and let $\mathcal{T}_h$ denote a finite element mesh. This mesh is made up of a finite number of subsets, κ, such that

(i) $\Omega = \bigcup_{\kappa \in \mathcal{T}_h} \kappa$,

(ii) For each distinct $\kappa \in \mathcal{T}_h$, it is assumed that the intersection of the interiors of each subset is equal to the empty set $\emptyset$.

These subsets, κ, are called the elements of the mesh. In $\mathbb{R}^1$, intervals are the main element type. When considering areas in $\mathbb{R}^2$ and surfaces in $\mathbb{R}^3$, meshes are made up of simple polygons, most commonly triangles and quadrilaterals. Typically, the elements that make up the mesh are all of the same type; however, hybrid meshes can also be used if required.

3.3.2 *Quadrature Methods*

In order to solve the integro-differential equations that arise in continuum models, a numerical quadrature rule is required to evaluate the integral. As introduced in Section 2.5, it is convenient to define these rules on a local reference element, which we define as $\hat{\kappa}$, and then use a mapping to perform the quadrature on any global element, similarly named $\kappa \in \mathcal{T}_h$. For a reference element $\hat{\kappa}$, the integral of a function, g, on that element is approximated by

$$\int_{\hat{\kappa}} g\left(\boldsymbol{\xi}\right) \mathrm{d}\boldsymbol{\xi} \approx \sum_{q=1}^{N_q} g\left(\boldsymbol{\xi}_q\right) \rho_q, \tag{65}$$

where $\boldsymbol{\xi}$ is the local coordinate system spanning $\hat{\kappa}$, $\{\boldsymbol{\xi}_q\}_{q=1}^{N_q}$ is a set of N_q quadrature points on $\hat{\kappa}$ and $\{\rho_q\}_{q=1}^{N_q}$ is a corresponding set of weights. Exploiting a mapping $\varphi_\kappa : \hat{\kappa} \mapsto \kappa$, this rule is written for a general element κ as

$$\int_{\kappa} g\left(\boldsymbol{x}\right) \mathrm{d}\boldsymbol{x} = \int_{\hat{\kappa}} g\left(\varphi_\kappa\left(\boldsymbol{\xi}\right)\right) \left|J_\kappa\left(\boldsymbol{\xi}\right)\right| \mathrm{d}\boldsymbol{\xi} \approx \sum_{q=1}^{N_q} g\left(\varphi_\kappa\left(\boldsymbol{\xi}_q\right)\right) \left|J_\kappa\left(\boldsymbol{\xi}_q\right)\right| \rho_q, \tag{66}$$

where $\boldsymbol{x}$ is the global coordinate system and $\left|J_\kappa\left(\boldsymbol{\xi}\right)\right|$ is the Jacobian of the mapping φ_κ. To approximate the integral across the whole domain we take the sum of the integral approximations across all elements, giving

$$\int_{\Omega} g\left(\boldsymbol{x}\right) \mathrm{d}\boldsymbol{x} = \sum_{\kappa \in \mathcal{T}_h} \int_{\kappa} g\left(\boldsymbol{x}\right) \mathrm{d}\boldsymbol{x} \approx \sum_{\kappa \in \mathcal{T}_h} \sum_{q=1}^{N_q} g\left(\varphi_\kappa\left(\boldsymbol{\xi}_q\right)\right) \left|J_\kappa\left(\boldsymbol{\xi}_q\right)\right| \rho_q. \tag{67}$$

This is a generic rule for all element types. As an example, consider a triangulated domain in $\mathbb{R}^2$. A common local reference element for triangular elements is the triangle defined by $\widehat{T} = \{\boldsymbol{\xi} \in \mathbb{R}^2 : \ 0 < \xi_1, \xi_2 \quad \text{and} \quad \xi_1 + \xi_2 < 1\}$. It can be shown that the Jacobian of the mapping between this reference element and any global traingle is equal to two times the area of the global triangle [8, 49]. Therefore, a rule can be written for a domain in $\mathbb{R}^2$, made up of triangular elements, as

$$\int_{\Omega} g\left(\boldsymbol{x}\right) \mathrm{d}\boldsymbol{x} \approx \sum_{\kappa \in \mathcal{T}_h} \left(2A_\kappa \sum_{q=1}^{N_q} g\left(\varphi_\kappa\left(\boldsymbol{\xi}_q\right)\right) \rho_q \right), \tag{68}$$

where A_κ denotes the area of the global element κ. This rule follows analogously for triangulated surfaces in $\mathbb{R}^3$.

3.3.3 *Gauss Quadrature on Intervals*

For the standard quadrature interval, $[-1, 1]$, the quadrature rule is given by

$$\int_{-1}^{1} g(\xi) dx \approx \sum_{i=1}^{N} w_i g(\xi_i). \tag{69}$$

When extending this to a general interval $[a, b]$, we must employ a mapping to go between local and global elements. With local coordinates, $\xi \in [-1,]$, and global coordinates, $x \in [a, b]$, this mapping is defined as

$$x = a + \frac{b-a}{2}(\xi + 1). \tag{70}$$

Finally, in order to change the variable of integration we require the Jacobian of the mapping, defined by $J_{ij} = \frac{\partial x_i}{\partial \xi_j}$, which in this case is

$$J(\xi) = \left(\frac{b-a}{2} \right). \tag{71}$$

Now, we have

$$\int_a^b g(x) dx = \int_{-1}^{1} g\left(x(\xi)\right) |J(\xi)| \, d\xi = \frac{b-a}{2} \int_{-1}^{1} g\left(a + \frac{b-a}{2}(\xi + 1) \right) d\xi. \tag{72}$$

Therefore, we can write

$$\int_a^b g(x) dx \approx \frac{b-a}{2} \sum_{i=1}^{N} w_i g\left(a + \frac{b-a}{2}(\xi_i + 1) \right). \tag{73}$$

Using the points and weights introduced in Section 2.5.2, this is the Gauss-Legendre quadrature rule for a general interval $[a, b]$. It yields an exact result for polynomials of degree $2N - 1$.

3.3.4 *Spatial Discretisation of the Neural Field Equation*

The numerical solution of neural field equations is one of the primary focuses of this thesis. Here, we show how the techniques presented thus

far are used to spatially discretise a standard NFE (Eq. (33)) and turn it into a system of time-dependent ODEs, ready to be numerically solved.

Assume the domain, Ω, has been partitioned into a finite element mesh, $\mathcal{T}_h$. Define a set of nodes $\{\varphi_\kappa(\boldsymbol{\xi}_q),\ \kappa \in \mathcal{T}_h, q = 1, \ldots, N_q\}$ and let us collectively refer to these nodes as $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$. Note that $n \leq N_\kappa N_q$, where N_κ is the total number of elements in the mesh, due to the possibility of quadrature nodes being shared by elements. Introducing $u_h(\boldsymbol{x}, t)$ to denote the solution to the discretised problem, we approximate Eq. (33) by first applying a generic quadrature rule (Eq. (67)) to the integral term, giving

$$\begin{aligned}\frac{\partial}{\partial t}u_h(\boldsymbol{x},t) &= -u_h(\boldsymbol{x},t) \\ &\quad + \sum_{\kappa\in\mathcal{T}_h}\sum_{q=1}^{N_q} w\left(\boldsymbol{x},\varphi_\kappa(\boldsymbol{\xi}_q)\right) f\left(u_h\left(\varphi_\kappa(\boldsymbol{\xi}_q),t\right)\right)\left|J_\kappa(\boldsymbol{\xi}_q)\right|\rho_q + I(\boldsymbol{x},t) \\ &= -u_h(\boldsymbol{x},t) + \sum_{j=1}^{n} w(\boldsymbol{x},\boldsymbol{x}_j)\, f\left(u(\boldsymbol{x}_j,t)\right)\sigma_j + I(\boldsymbol{x},t), \end{aligned} \tag{74}$$

where

$$\sigma_j = \sum_{\{\kappa,q\,:\,\varphi_\kappa(\boldsymbol{\xi}_q)=\boldsymbol{x}_j\}} \left|J_\kappa(\boldsymbol{\xi}_q)\right|\rho_q. \tag{75}$$

Demanding that Eq. (74) be satisfied at the quadrature nodes $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$, this gives the system of ordinary differential equations (ODEs)

$$\frac{\mathrm{d}}{\mathrm{d}t}u_h(\boldsymbol{x}_i,t) = -u_h(\boldsymbol{x}_i,t) + \sum_{j=1}^{n} w(\boldsymbol{x}_i,\boldsymbol{x}_j)\, f\left(u_h(\boldsymbol{x}_j,t)\right)\sigma_j + I(\boldsymbol{x}_i,t), \tag{76}$$

for $i = 1, \ldots, n$, or in vector form

$$\dot{\boldsymbol{u}}(t) = -\boldsymbol{u}(t) + \boldsymbol{W}\cdot\boldsymbol{f}(\boldsymbol{u}(t)) + \boldsymbol{I}(t), \tag{77}$$

where $\dot{}$ represents the derivative with respect to time, $\boldsymbol{u} = (u_1, \ldots, u_n)$, $\boldsymbol{f} = (f_1, \ldots, f_n)$, $\boldsymbol{I} = (I_1, \ldots, I_n)$, $u_i(t) = u_h(\boldsymbol{x}_i, t)$, $W_{ij} = w(\boldsymbol{x}_i, \boldsymbol{x}_j)\,\sigma_j$, $f_i(\boldsymbol{u}) = f(u_i)$, and $I_i = I(\boldsymbol{x}_i, t)$.

This type of spatial discretisation for integral equations is called the

Nyström method [130, 8, 117]. We employ it throughout this thesis to discretise all the contintuum models of neural activity that we explore. A one dimensional example of a mesh discretisation that can be applied in this way is shown in Figure 5. Here, we divide the domain $\Omega = [-3, 3]$ into a set of interval elements of uniform width $h = 2$ and subsequently apply a $N_q = 3$ point Gauss quadrature rule to each element. The blue points represent the uniform grid points that make up each element and the red points represent the Gauss points on each element.

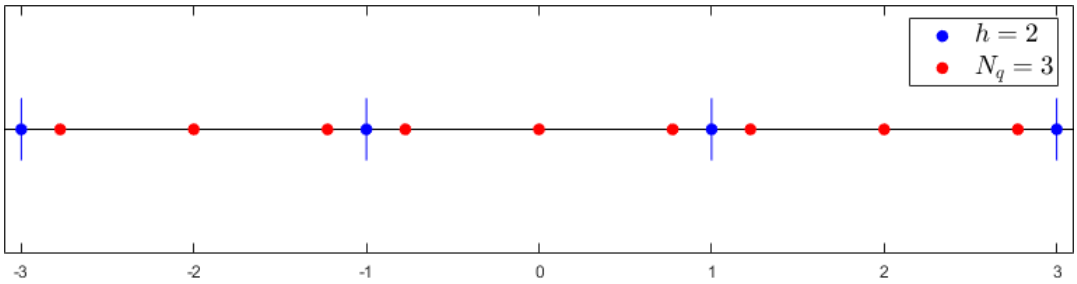


Figure 5: Example one dimensional mesh discretisation on $\Omega = [-3, 3]$, divided into interval elements of width $h = 2$ (blue dots), and a $N_q = 3$ point Gauss quadrature rule applied to each element (red dots).

The system of equations Eq. (77) can be solved using a timestepper in order to achieve a full spatio-temporal approximation to Eq. (33). For this approach, computation time is heavily dependent on the sparsity of W due to the matrix-vector multiplication step in Eq. (77).

3.3.5 *Error Analysis*

In this section we discuss the order of convergence of the numerical methods presented in Section 3.3.4 and undertake a series of convergence tests to verify that the results of the simulated discretised neural field equation conform to theoretical expectations.

Let $u(\boldsymbol{x}, t)$ be a solution to Eq. (33), and let $u_h(\boldsymbol{x}, t)$ be a solution to the discretised problem Eq. (77), where $u_i(t) = u_h(\boldsymbol{x}_i, t)$. To measure the error between the discrete solution $u_h(\boldsymbol{x}, t)$ and the true solution $u(\boldsymbol{x}, t)$, we compute $\|u - u_h\|$, where $\|\cdot\|$ is a norm of our choosing. For this section, we consider cases where the true solution $u(\boldsymbol{x}, t)$ is available in closed form. Several examples can be found in [117] for addressing error in both the spatial and temporal discretisations. The timestepper used in this section is MATLAB's ode45 solver [153], which is a sufficiently accurate temporal solver that the spatial error should dominate in the solution enough to observe the convergence. For the one dimensional case, we use an example posed on a domain $\Omega = [-L, L] \subset \mathbb{R}$. It can be shown that for a firing rate as seen in Eq. (34), a connectivity kernel

$$w(x, x') = w(x - x') = \mathrm{e}^{-(x-x')^2}, \tag{78}$$

and external input

$$\begin{aligned} I(x,t) = \theta &- \left(\frac{\gamma \exp(\gamma t + x^2)}{\mu(\exp(\gamma t + x^2) - G)} \right) - \frac{1}{\mu} \log \left(\frac{\exp(\gamma t + x^2)}{G} - 1 \right) \\ &- \frac{1}{2}\sqrt{\frac{\pi}{2}} G \exp\left(-\gamma t - \frac{x^2}{2} \right) \left(\operatorname{erf}\left(\frac{2L - x}{\sqrt{2}} \right) + \operatorname{erf}\left(\frac{2L + x}{\sqrt{2}} \right) \right), \end{aligned} \tag{79}$$

where $0 < G < 1$ and $\gamma > 0$ are positive real constants, that the analytical solution is given by

$$u(x,t) = \theta - \frac{1}{\mu} \log \left(\frac{\exp(\gamma t + x^2)}{G} - 1 \right). \tag{80}$$

This solution is derived by forcing the integral term of the NFE to have an analytical solution by carefully selecting the integrand, and then working backwards to yield the closed-form solution, u, and external input term, I. For the one dimensional Nyström discretisation, as explained in Section 3.3.4, uniform interval elements of width h are used in conjunction with an N_q point Gauss quadrature rule. In order to produce error plots to illustrate the convergence of the method, we select the infinity norm of $u - u_h$ as the chosen error measure. This is defined by

$$\|u - u_h\|_\infty \approx \max_{\{\boldsymbol{x}_i,\ i=1,\ldots,n\}} \max_{\{t_s,\ s=0,\ldots,m\}} \left| u(\boldsymbol{x}_i, t_s) - u_h(\boldsymbol{x}_i, t_s) \right|, \tag{81}$$

where t_s, $s = 1, \ldots, m$, denotes the temporal discretisation points.

In [117], it is determined that for sufficiently smooth u, w, and f, the quadrature method should converge with order $\mathcal{O}\left(h^{2N_q}\right)$. Evidence of this can be seen in the error plot depicted in Figure 6. The error for the $N_q = 2$ rule (red) is clearly of order $\mathcal{O}\left(h^4\right)$, as h tends to zero. For the $N_q = 3$ rule (black), there are several points which fit with the $\mathcal{O}\left(h^6\right)$ line before the timestepper error dominates, and for the error in the $N_q = 4$ rule (blue), the timestepper error dominates before any points can fit the $\mathcal{O}\left(h^8\right)$ line. The pre-asymptotic region, where h is comparatively large, appears to show convergence at a faster rate.

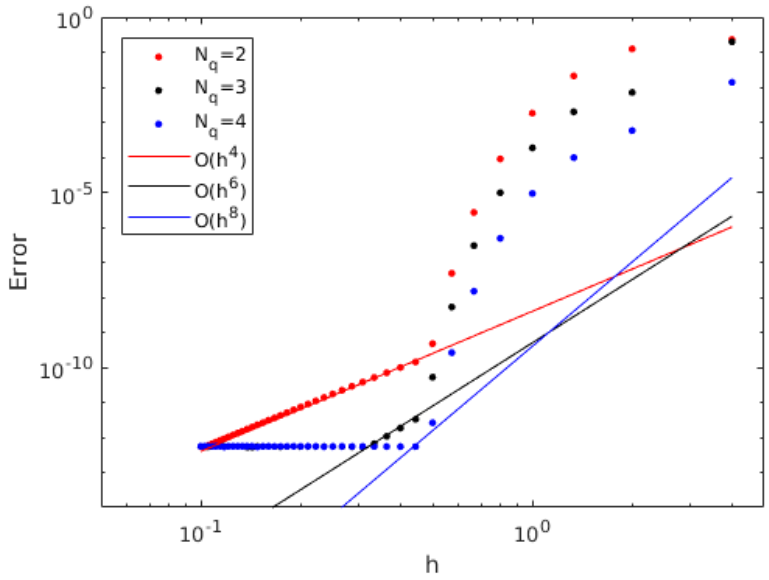


Figure 6: Error plotted against uniform step size h for varying N_q point Gauss quadrature rules posed on the domain $\Omega = [-4, 4]$. Initial condition specified as the analytic solution given by Eq. (80) at $u(x, 0)$.

In Figure 7, the error plot for a two dimensional example taken from [117] is shown. This is posed on the domain $\Omega = [-L, L]^2 \subset \mathbb{R}^2$ and uses the firing rate $f(u) = \tanh(u)$, connectivity kernel

$$w(\boldsymbol{x}, \boldsymbol{x}') = w(x_1, x_2, x_1', x_2') = \exp\left(-(x_1 - x_1')^2 - (x_2 - x_2')^2\right), \tag{82}$$

and external input $I(\boldsymbol{x}, t) = I(x_1, x_2, t)$, where

$$I(x_1, x_2, t) = 1 + t - \frac{\pi}{4}\tanh(t)\left(\operatorname{erf}(1 - x_1) + \operatorname{erf}(1 + x_1)\right)\left(\operatorname{erf}(1 - x_2) + \operatorname{erf}(1 + x_2)\right). \tag{83}$$

The analytical solution is $u(\boldsymbol{x}, t) = t$. This example is selected due to the timestepper being able to solve it exactly in time, therefore only the error in the spatial discretisation is present.

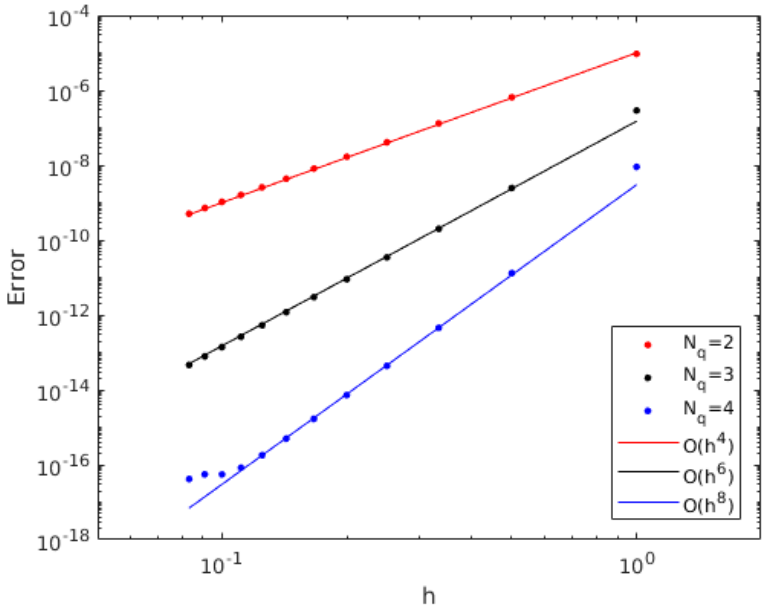


Figure 7: Error analysis example from [117] posed on the domain $\Omega = [-1, 1]^2$, for the case where there is no temporal discretisation error. Initial condition specified as $u(\boldsymbol{x}, 0) = 0$.

For this example, it is clear that the quadrature error is of order $\mathcal{O}\left(h^{2N_q}\right)$, as expected. The tailing phenomenon seen in the blue dots when they reach $\approx 10^{-16}$ is due to machine tolerance error taking over and polluting the

solution. This is due to computers only storing floating point numbers to a specific level of precision.
In [11], this error analysis example is reproduced in both one and two dimensions with a variety of Nyström discretisations.

3.4 SUMMARY

This chapter has outlined some of the analytical and numerical techniques most commonly employed to aid in the study of models of neural activity. We began by demonstrating how an arbitrary non-linear network of neurons can be constructed. This work, along with the corresponding linear stability analysis of the network model, is important as a prerequisite to the work undertaken in Chapter 6. Following on from this, we discussed the process of spatially discretising continuum models of neural activity so that they may subsequently be evolved temporally via numerical differential equation solution algorithms, which we will employ in Chapters 5 and 7. This included introducing the quadrature methods required to approximate the spatial integral term of neural field models, and specifically defining a Gauss quadrature rule for interval domains. Finally, an error analysis was undertaken to verify that the discretisation and quadrature schemes yield results that conform to the theoretical expectations.

4

NFESOLVE: AN OBJECT-ORIENTED DIFFERENTIAL EQUATION SOLVER

4.1 INTRODUCTION

One of the main objectives of this thesis is to provide an enhanced suite of numerical differential equation solvers to brain modelling researchers. The primary motivation behind the development of this suite is to aid in the solution of non-local and nonlinear neural field equations with multiple delays. Although there are several pre-existing solvers written in a variety of languages, we found that when using these solvers for solving equations on a large-scale mesh with many delays, the computation time and memory requirements were extremely high, often requiring a powerful server to be run on. This posed the question as to whether a new, more efficient solver could be developed that specifically caters for the needs of large-scale delayed neural field models.

Others have taken a variety of different approaches, such as Hutt et al. [91] who use their own scheme to discretise the NFE and diagonalise the delay history, making it easy to access the specific history state when needed. They then use a forward Euler time-stepper to generate the solution. Venkov [175] builds upon this work and further improves the scheme by using a differing set up of the diagonalised history array and changing the integration rule. They also employ the use of MATLAB's dde23 solver

and state that their scheme achieves a speed up of approximately one third faster than that of Hutt et al. [91]. However, the scheme used is very rigid in its restrictions on the spatial and temporal step sizes in order to ensure that the required delay states at each step coincide perfectly with the already computed solution states. There is no scope for adaptive stepping and a lot of preprocessing is required on the format of the problem. Faye and Faugeras [61] present both functional and numerical analysis results for the convergence of their own discretisation scheme; however, they too make use of MATLAB's dde23 routine for the time-stepping. Visser et al. [176, 177] have developed their own scheme which makes use of cubic Hermite splines to interpolate between history states. They formulate the discretised problem into a large matrix-vector system using basis matrices to store all the interpolatory coefficients. Once again, a forward Euler time-stepper is employed due to the rigidity of the step size restrictions not allowing for methods, such as Runge-Kutta, that require intermediary calculations at fractional values of the time-step. We aim to eliminate the need for restrictive conditions and allow the user to flexibly solve neural field models in an efficient and timely manner.

Perhaps the closest existing entity to what we wish to achieve is The Virtual Brain project [148]. This contains a wide selection of tools, written in Python [170], specifically for the simulation of network dynamics and the visualisation of brain data. Evidently, this codebase has been in production for a number of years and has had a variety of different contributors, therefore the full range of features available is far more extensive than what we are proposing the initial release of our library to contain. However, this library focus heavily on the solution to stochastic differential equations (SDEs), with a particular application to neural mass models, whereas the suite we propose casts more of a spotlight on delay differential equations (DDEs) and their relationship to neural field models. Although The Virtual Brain does contain some DDE solvers, the algorithms behind them are limited in their flexibility; for example, the step size must

remain fixed throughout. The choice of programming language is also important. Python is a powerful programming language used in a plethora of applications; however, as it is a dynamically typed interpreted language it does not always perform to the same standard as a statically typed compiled language, such as C++ [162] or Java [5].

In this chapter, we detail the design choices made in the development of this software package, along with a description of the components that it consists of and the differences and similarities to other more conventional solvers. Outlined first are the preliminary measures that were considered before development began. Following this, explanations of the classes that make up the software are provided, split into four main groups: mesh handler, ODE solvers, DDE solvers and sparse DDE solvers.

4.2 PRELIMINARIES

Performance is one of the primary elements that we wish to champion in this suite of codes, hence the need for careful consideration of the underlying mechanisms that the code is built upon. Here, 'performance' refers to both acheiving low computation times and reducing hardware requirements, such as RAM. The first step in the development of this package was to choose an appropriate coding language to develop in. As discussed in the previous section, the choice of programming language is one of the most important elements when developing code. After some consideration of the languages available, we determined that C++ met all of the requirements that were needed by the package. C++ is a mid-level programming language that allows for very fast computations and the use of object-oriented design principles. It is widely regarded as one of the most powerful languages and is used in many applications, from game development to operating systems to intense numerical simulations. The chosen standard selected here is C++11 [96].

Once a programming language was selected, the next task was to determine an appropriate library of optimised data structures, such as vectors and matrices, to utilise in development. This was a very important consideration as the default array system in C++ does not have any operations defined and is very difficult to work with in a numerical setting. The Armadillo library [145, 146, 147] is a high quality linear algebra library that provides efficient vector and matrix classes with a MATLAB-like syntax, along with sophisticated numerical algorithms and operations. It uses BLAS (Basic Linear Algebra Subprograms) [19] and LAPACK (Linear Algebra Package) [3] to provide the linear algebra routines, and also has the capabilities to automatically use OpenMP [43] parallelisation to speed up intensive computations. Additionally, sparse matrix structures are supported and have greatly optimised operations, decompositions and matrix algebra solvers. Armadillo is currently used across a number of existing codebases, with a wide range of applications. This makes it a very strong choice for the development of this package. Here, we employ the OpenBLAS [183] implementation for the back-end as it is a highly optimised version of BLAS, providing much faster matrix algebra routines.

The entire suite of code, along with a selection of example problems, is stored on a GitHub repository and is available to download at `https://github.com/UoN-Math-Neuro/NFESOLVE`.

4.3 MESH HANDLER DESIGN

As part of the NFESOLVE package, a mesh handler toolkit is included. This provides a base class to store details of a geometric mesh, along with sub-classes for specific mesh types, and a quadrature library to generate the weights and points associated with various quadrature rules depending on the element type and dimension of the geometry. Although these classes can

be applied in many different settings and to a wide selection of problems, the main motivation behind developing them as part of the NFESOLVE library was to manage the spatial integrals that are typically part of neural field models.

4.3.1 *Mesh Class and Subclasses*

The *Mesh* class is a base class designed to store the coordinates and element data that make up a finite element mesh. Thus far, it is built to handle non-hybrid polygonal meshes in any given dimension. The majority of the methods in this class are 'getter' methods to return the attributes that the class holds. These include custom methods for returning specific individual element connectivity arrays and grid point coordinates. If the user at any point wants to change the grid points or connectivity data without instantiating a new object, the class provides 'setter' methods to do so. Finally, there is an output method that allows the mesh data to be written to a file so that, if the user requires, the mesh can be utilised outside of the scope of the NFESOLVE library. We have designed our own custom data file structure to facilitate the writing out of this data.

Mesh Data File Format

```
DIMENSION OF MESH (int)
NUMBER OF GRID POINTS (int)
NUMBER OF ELEMENT VERTICES (int)
NUMBER OF ELEMENTS (int)

LIST OF GRID POINT COORDINATES (double)

LIST OF ELEMENT CONNECTIVITY ARRAYS (int)
```

This data file is a file of type `.dat`. The first two entries dictate the number of columns and rows of the grid point coordinates matrix, respectively, and the following two entries give the number of columns and rows of the element connectivity arrays matrix, respectively. This mesh file format is also required by the *MeshFromFile* subclass in order to read in meshes from external sources. Users are expected to formulate their mesh data into a

data file before hand, assuring that it is structured in this specific format.

For basic meshes in lower dimensions, the user may not have a data file readily available and would be required to generate this file before they could proceed. In order to provide a good user experience, we have supplied a selection of other subclasses to the *Mesh* class which can be used to automatically generate standard meshes for simple domains, such as intervals and rectangles, which are the shapes of choice for many idealised studies.

The first of these, namely *Regular1DGrid*, has two possible constructors. The first generates a linearly spaced 1D grid in a similar way to the `linspace()` function found in languages such as MATLAB and Python. Users supply a start and end point along with the number of points they would like the grid to have. The grid points and element connectivity data are then generated automatically. Alternatively, the user can supply a pre-constructed vector of grid points (without regard to whether they are uniformly spaced) and the constructor will build the element connectivity array for those given grid points.

Two other standard classes that are provided are *Regular2DTriGrid* and *Regular2DQuadGrid*. These provide discretisations of rectangular domains, using triangular and quadrilateral elements respectively. They each have three possible constructors but these have the same arguments for both element types. Similarly to the *Regular1DGrid* class, the first two of these allows the user to specify the start and end points for both vertical and horizontal directions. There is then the option to input either one or two values dictating the number of grid points in each direction. If only one value is entered then there will be the same number of grid points in each direction. Using these inputs, a rectangular lattice of grid points is formulated and the element connectivity array is generated based on which element type is required. The third type of constructor takes in an array of pre-defined grid points for each direction and then builds the

rectangular lattice and element connectivity array from the supplied points. For the quadrilateral grid, each grid point is connected to its neighbouring grid points, thereby creating rectangular elements. The triangulated grid is similar to the quadrilateral grid; however, each rectangle is divided diagonally in half from bottom right to top left, thus generating triangular elements.

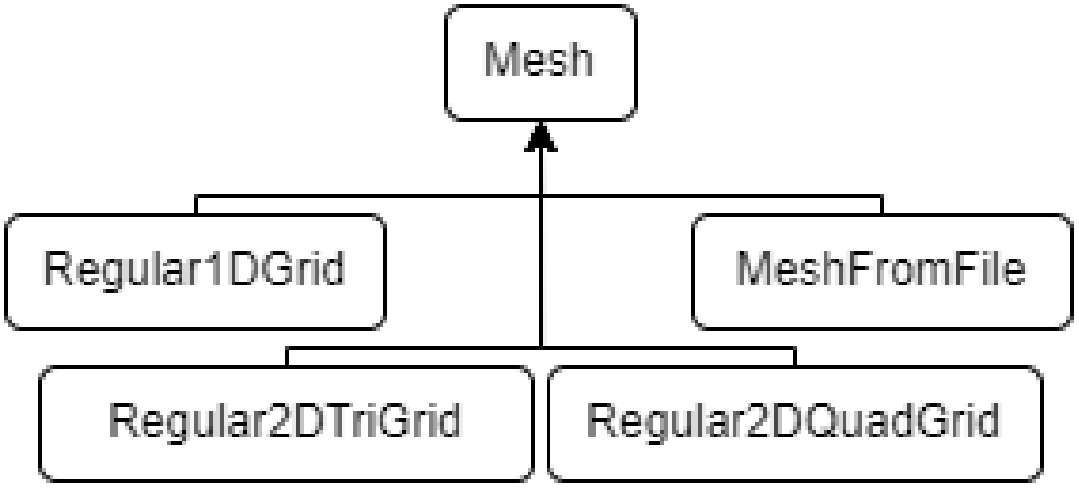


Figure 8: Diagram illustrating the inheritance-based structure of the mesh classes.

The inheritance tree of the mesh classes is shown above in Figure 8.

4.3.2 *QuadratureLibrary Namespace*

Another key part of the NFESOLVE library, that goes hand in hand with the mesh classes (Section 7.2.1), is a suite of quadrature rules that allow for numerical integration to be performed over a mesh. This is particularly relevant to neural field models due to the spatial integral term that is typically present. This is designed in the form of a namespace instead of a class. A namespace in C++ is the equivalent of a class with static methods. These are methods that do not require an object of the class to be instantiated before being used.

Thus far, the *QuadratureLibrary* namespace contains methods for generat-

ing the quadrature weights associated with interval, triangular and regular quadrilateral element types. These methods all take in an object of type *Mesh*, which contains the relevant information about the element type and dimension, allowing for the appropriate rule to be applied. The simplest of the rules in the suite are those that only utilise the vertices of the elements. These rules do not require any additional quadrature nodes to be generated, and for higher dimensional meshes correspond to the trapezoidal rule in one dimension. They can therefore be applied directly to the original meshes and can be very useful in providing an efficient approximation to an integral where integrand values are only known at the supplied mesh points, such as when using real-world data. For interval domains, a method to generate the weights for Simpsons's rule has also been implemented. As detailed in Section 8.2, future considerations include developing a vertex rule for irregular quadrilateral elements and adding functions for more accurate methods, such as Gauss quadrature schemes, on all element types.

4.4 ODE SOLVER DESIGN

Included in the codebase are a series of classes that are designed to facilitate the numerical solution of ODEs. In order to keep the code as flexible as possible, a polymorphic design structure is used. This relies on a base interface class, named *ODEInterface*, which provides a structure for defining a specific problem to be solved. This is an important design decision as it allows for the solver classes to accept any ODE problem that is built from this interface. The first consideration when building the solvers was that, although numerical ODE solvers are mostly distinct in their properties when compared to other types of differential equation solvers, there are still several aspects which are shared commonly, e.g. a state vector. We make use of an abstract class, named *AbstractDESolver*, to handle the core qualities that all types of differential equations share. Deriving from this, we employ a *AbstractODESolver* class to act as a base for different numerical ODE

solvers. This contains the properties that are unique to just ODE solvers. At the time of writing, the specific solution algorithms that have been implemented as part of the NFESOLVE library include both a 3rd and 4th order fixed-step Runge-Kutta scheme, along with an adaptive-step 3(2) order Runge-Kutta scheme (see Appendix A.1.3 for details on embedded Runge-Kutta schemes). The framework that these are built upon allows for other solvers to be easily implemented in the future if they are required, such as higher or lower order Runge-Kutta methods and stiff equation solvers.

4.4.1 *ODEInterface Class*

In order to solve a given ODE, a solver class needs to be able to accept an object that contains all the relevant information pertaining to the desired problem. However, every ODE is different and every user's requirements to set up their specific problem will likely be unique to the problem they are trying to solve. We choose to combat this by formulating a base interface class, named *ODEInterface*, which contains common attributes that all ODE problems share, such as a 'right-hand-side' (or 'F'). As this is an interface, it will never be directly instantiated, but it does allow for a solver class to expect an object that follows a certain contract and for it to call any relevant methods consistently across all problems. Users are able to add in any additional information or processing steps that they see fit into their derived class, meaning key components such as model parameters or extra function evaluations can be incorporated, the only requirement is that they implement any pure virtual methods defined in the *ODEInterface* class. A pure virtual method is a method that has no defined implementation and must be overridden by any derived class that is itself not abstract.

To illustrate the design of this interface, we use the typical form of an ODE, given by

$$y'(t) = F(t, y(t)), \qquad y(t_0) = y_0. \tag{84}$$

For simplicity, we refer to the variable t as representing 'time'. The *ODEInterface* class contains a pure virtual method for computing the right-hand-side, named `ComputeF()`, which is built to take in a time, t, and a state vector, y, as input arguments. This is pure virtual as there is no generic implementation; it must always be defined in a derived class pertaining to the specific problem. There also exists a virtual method for computing the analytic solution at a given time point, if the analytic solution is known. This typically will not be used, hence why it is just a virtual method and not a pure virtual method; however, it is included so that it can be used if required. An example where this method would be useful is for processes such as numerical error analyses. It is especially important for the user to make sure that the contract for these methods always remains consistent across all derived classes that they create, as the solvers which utilise the interface are only aware of the methods and variables that are defined in the interface, not any other methods or variables which the user may implement within their child classes.

4.4.2 *AbstractDESolver Class*

Similarly to the *ODEInterface* design, we wish to employ an inheritance-based architecture for the suite of differential equation solvers. Before moving directly to discussing the ODE solvers, it is pertinent to discuss the commonalities that arise across numerical solvers for all forms of differential equation (ODEs, DDEs, etc.). Although the numerical algorithms for each type of differential equation can vary in a multitude of ways specific to the type of problem that is being solved, there are a number of common properties that are shared amongst all types of solvers

which can be abstracted out into a parent class that acts as a base for all types of numerical differential equation solver.

Firstly, no matter which type of equation is under scrutiny, there will also be an initial and final time point that the solver is instructed to compute the solution between. Another key aspect that is shared across all numerical differential equation solvers is the current step size with which to integrate forward. Whether this has a fixed value or it adapts according to the solution, it is not distinct to any type of problem and can therefore be defined outside of any specific solver classes. Most importantly, the main piece of information that is required by any numerical solver is the current solution state of the system. Following the design principles outlined thus far, we supply the parent class, *AbstractDESolver*, to hold these attributes. In addition to the properties described above, we also opt to include a number of variables for facilitating the output of the current solution state so that the user can access it externally to the application. These include the name of the output file that the solution will be saved to, an array containing the indices of the solution state that are to be saved (in case the user does not wish to unnecessarily store data that is not of interest to them), a 'save gap' variable which determines how frequently the stepped solution will be saved to the output file, and a 'print gap' variable to specify the frequency at which the solution should be printed to the console (if the user wishes to visualise the numerical data in real time).

In terms of the methods that are available in this abstract class, the main one of interest is a pure virtual method named `Solve()`. In all of the derived solver classes, this is where the specific solution algorithms will be implemented. We choose to declare this method in this parent class to keep the method name reserved for implementation. As is standard, 'setter' and 'getter' methods are provided for all the variables listed, allowing for multiple different simulations to be run back-to-back from the same solver

object. Additionally, this parent class contains two non-virtual methods which format and then print a given solution state to the console or save it to a data file, while respecting the previously discussed 'print gap' and 'save gap' variables. Finally, a general method to print an elapsed time frame to the console is also included. This can, for example, be used within the child solver classes to display the time taken for the `Solve()` method to complete.

4.4.3 *AbstractODESolver Class*

Although the *AbstractDESolver* class handles the key properties that are shared across all numerical differential equation solvers, it does not account for anything specific to ODEs. Following a similar approach to the *AbstractDESolver* design, a new abstract class that inherits from this parent class is provided to cater for ODE problems. This class is called *AbstractODESolver*. As it inherits from *AbstractDESolver*, all the variables and methods discussed in Section 4.4.2 are available to access. The main distinguishing feature that gears this class specifically towards ODE problems is that it stores a pointer of type *ODEInterface* that will be used in a polymorphic way to access the specific ODE problem that is being solved. If the user wishes at any point to change the ODE which is being pointed to but maintain the solver parameters defined within a solver object, a setter method has been provided as part of this class. Any classes which derive from *AbstractODESolver* are now fully equipped with access to everything they need to implement a numerical ODE solution algorithm. This design structure means that new solvers can be added in to the NFESOLVE library at any point without breaking existing code and while maintaining uniformity across all solver structures.

4.4.4 *RungeKuttaSolver Classes*

One of the most widely used algorithms for numerically solving differential equations is the Runge-Kutta scheme [109]. Provided with the NFESOLVE package are a selection of Runge-Kutta solvers for ODEs. As explained in Appendix A.1, the Runge-Kutta scheme is generalised to have s stages which are defined by the Butcher tableau coefficients a_{ij}, b_i, and c_i, for $i = 1, 2, \ldots, s$ and $j = 1, 2, \ldots, i-1$ in the case of explicit methods. These coefficients dictate the order of accuracy at which the method performs. In order to facilitate generating Runge-Kutta classes of any specific order, a key design choice is to implement a base class, named *RungeKuttaSolver*, which derives from *AbstractODESolver* and contains methods for both the non-adaptive and adaptive Runge-Kutta algorithms. These methods take in the Butcher tableu coefficients as input arguments, allowing for any Runge-Kutta scheme to be utilised. Before the actual implementation of the algorithm, the methods begin by printing a header to the console which details all the information about the chosen solver and the chosen parameters. A clock is started to monitor the elapsed wall time (real-world time taken to complete a task) for the duration of the stepping process. At the end of each computed step, a check is made to determine if the step count is divisible by either the print gap or save gap variable. If the check returns true then the corresponding output method (defined in *AbstractDESolver*) is called. The stepping algorithm proceeds to the next step and this process is repeated until the time variable reaches the final time that was specified by the user. Finally, the total elapsed wall time is printed to the screen. As these algorithms are only implemented once, instead of repeatedly for each individual Runge-Kutta solver, this allows for any future changes or bug fixes to apply globally to all specific Runge-Kutta solvers. To implement a specific Runge-Kutta method, we simply create a new class that derives from the *RungeKuttaSolver* class and override the pure virtual `Solve()` method (defined in *AbstractDESolver*) to

call either the non-adaptive or adaptive Runge-Kutta algorithm with the desired coefficients.

Thus far, there are two fixed-step explicit Runge-Kutta solvers and one adaptive-step explicit Runge-Kutta solver that have been developed and tested. The first of the three implemented solvers is a 3rd order method, known as Kutta's method. This is defined in a class named *RungeKutta3Solver*. Internally, pre-defined private variables contain the relevant coefficients for this method. When an object of type *RungeKutta3Solver* is instantiated, the constructor requires the user to supply a reference to a child object of *ODEInterface*, an initial condition, the initial and final times that the solution is required between, the step size they wish to be used, a name for an output file, a 'save gap', and a 'print gap'. Optionally, the user may also supply a vector containing a list of indices of the state variables they wish to be outputted. If this is not specified, the default implementation is to output every element of the state vector. To begin the stepping process, the user simply calls the `Solve()` method, which as explained above, calls the Runge-Kutta implementation method with the defined coefficients.

The second of the fixed-step solvers included in the NFESOLVE library is a 4th order method, defined by the classical Runge-Kutta formula [109]. This class is named *RungeKutta4Solver* and is implemented analogously to the *RungeKutta3Solver* class, with the appropriate adjustments to the privately-defined coefficients.

The third and final ODE solver that has been implemented to date is an adaptive-step scheme that uses the 3rd order Runge-Kutta method detailed above for the main algorithm. The step size adaptation is determined by using this approximation in conjunction with an embedded 2nd order Runge-Kutta method whose coefficients are computed based on the 3rd order coefficients. The process behind the scheme embedding and step size adaptation is detailed in Appendices A.1.3 and A.1.4. This approach

is very similar to that used in MATLAB's ode23 solver [20]. We name this adaptive solver class *RungeKutta32Solver* as it is a 3(2) scheme. Instead of the constructor requiring the user to pass in a step size as an argument, they instead input their choice of the absolute and relative tolerances that comprise the adaptation error condition.

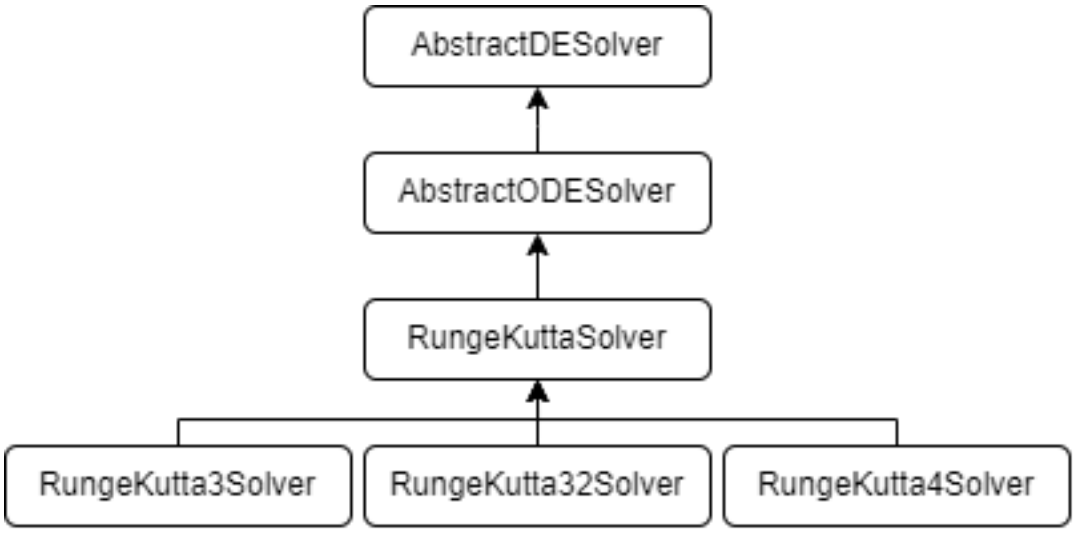


Figure 9: Diagram illustrating the inheritance-based structure of the ODE solver classes.

The diagram above depicts the structure that we have just described, showing the inheritance tree of the ODE solver classes.

As the code was designed in this modular way, there is scope for different solvers to easily be integrated into the package in the future. For the purposes of this project, only a selection of solvers that meet the needs of the neural field problems being solved have been implemented thus far. This is in order to balance computational efficiency and intensity with acceptable error margins.

4.5 DDE SOLVER DESIGN

The construction and implementation of a suite of classes to solve DDEs follows on from the ODE solution package by building upon and expanding the modular framework used. Developed alongside the standard DDE solvers are a set of solvers that make use of a sparse data structure to allow for certain DDE problems to be solved in a noticeably more efficient manner. To facilitate the development of these solvers, three additional classes are required. The first two of these provide a storage buffer for the solution history as it is stepped forward. One of these is static (size is fixed) and is utilised by the fixed-step solvers, while the other is dynamic (size can adapt) and is employed by the adaptive-step solvers. For the sparse solvers, a custom class named *SparseDelayMatrix* implements a sparse matrix structure, based on the compressed sparse column (CSS) storage format (see Appendix A.3.1), to efficiently store the matrix of delay states that is computed at each stage of the underlying Runge-Kutta algorithm.

Although there does exist methods for solving DDEs with time-dependent or state-dependent delays, considering the principal intended use of the software, we have opted to only implement algorithms for the solution of DDEs with constant delays, thus far.

4.5.1 *DDEInterface Class*

Analogously to the ODE section of the NFESOLVE library, in order to solve a particular DDE, an interface is required to provide a base for all DDE problems to derive from. This interface is then used by the solver classes so that they may accept any DDE problem and consistently call specifically reserved methods. The class is named *DDEInterface* and it mirrors the design aspects of the *ODEInterface* class.

As shown in Appendix A.2.1, for a series of m distinct delays, $\tau = \{\tau_1, \ldots, \tau_m\}$, the general form of a constant delay DDE is given by

$$y'(t) = F(t, y(t), y(t-\tau_1), \ldots, y(t-\tau_m)), \quad t \geq t_0, \tag{85}$$

$$y(t) = \varphi(t), \quad t \leq t_0, \tag{86}$$

where φ is a history function that defines the solution prior to the initial time, t_0. Instead of writing the right-hand-side as being a function of many delay states, it can be written succinctly as $F(t, y(t), Z(t))$, where $Z(t)$ is a matrix such that $Z(t) = [y(t-\tau_1), \ldots, y(t-\tau_m)]$. This allows for a much simpler formulation of the pure virtual method contract that is reserved to compute the right-hand-side of the problem, once again named `ComputeF()`, as only one method argument is required to encompass all the delay states. On top of the `ComputeF()` method, a new pure virtual method named `ComputeHistory()` is also required to evaluate the history function, $\varphi(t)$, when solution states that fall prior to t_0 are required. As all DDEs require a history, this method must be overridden in all child classes. Lastly, in conformity with the *ODEInterface* class, there is also a virtual method included to compute the analytic solution of the equation if it is known. This is very unlikely to be used, however, as it is a rare occurrence that DDEs present analytically solvable solutions.

4.5.2 *AbstractDDESolver Class*

Building upon the framework previously outlined, in order to begin the implementation of specific DDE solvers, it makes sense to define an abstract class that contains the common properties which all DDE solvers share. This class is called *AbstractDDESolver* and it derives from the general differential equation solver abstract class, *AbstractDESolver*. It is very similar to the *AbstractODESolver* class in many respects, except that instead of storing a

pointer to an object of type *ODEInterface*, it now contains storage for an object of type *DDEInterface*. There are also two new additions to the set of privately stored variables that are required in order to accommodate the delays. The first of these is a vector that stores all of the delay values present in the system. This is important as the solver will need to cycle through all the delays at each time-step and compute the relevant delay state vectors so that it can populate the Z matrix. Secondly, a simple integer dictating the number of equations in the system which actually require delay states to be computed is also included. An important design choice made here was to request that the user order their system by placing the equations which require delay states to be computed at the top of the state vector. This consideration was made to prevent unnecessary delay states being computed that would never be needed, therefore saving memory and computation time. Most pre-existing solvers do not take this into account and force delay states to be computed for the entire system, even if some of the equations in the system never actually use these delay states.

4.5.3 *HistoryBuffer Classes*

So as not to waste memory unnecessarily, the implementation of the ODE solver suite in the NFESOLVE library only stores the current state of the system as it is evolved. As the solution is outputted to a file (depending on the the 'save gap' the user has set), there is no need to store all the past solution states in memory, allowing resources to be reserved purely for the computation process. However, when it comes to DDEs, in order to progress the solution at any given time point, a solver must have access to the history of the solution so that it is able to compute the necessary delay states. If the stepper has not progressed past $t_0 + \tau_{\text{min}}$, where τ_{min} is the smallest delay value in the system, then all delay states are easily computed from the history function $\varphi(t)$. Once the stepper has moved beyond this value it will need access to the previously computed solution

states in order compute the delay states needed.
Although there exist algorithms, such as the 'Natural Runge-Kutta method for DDEs' (detailed in Appendix A.2.2), which are designed to allow the required delay states to coincide perfectly with the known solution states, in practice these schemes are typically non-viable due to the rigid restrictions imposed on the delays and solver step size. The aim of the NFESOLVE library is to remove as many restrictions as possible so that it may be used to efficiently solve all manner of models. One way of doing this is to make use of a 3rd order Hermite interpolant to compute delay states that fall in between the past solution states. The algorithms behind this interpolation method are explained in Appendices A.1.2 and A.2.3. The considerable upside to using an interpolant is that it allows for both non-adaptive and adaptive stepping, as opposed to rigidly fixing the step size such that all delay times fall perfectly on previously computed states. The chosen Hermite interpolant [151, 77] also only requires two neighbouring solution states for it to yield a 3rd order accurate approximation of the solution.

To utilise the interpolation scheme, solution states and their derivatives will need to be stored in memory instead of being forgotten about once the stepping algorithm moves on. However, storing every single computed solution is not necessary and would quickly saturate the available memory. All that is required by the solver is to have a history of solution states stored from the current time, t_n, back to $t_n - \tau_{\max}$ (similarly to $\tau_{\min}$, the largest delay value present in the system is represented by $\tau_{\max}$). To manage this, we formulate two new classes: one for the fixed-step solvers and one for the adaptive-step solvers. The first of these is named *StaticHistoryBuffer* and, as the name suggests, does not change in size. The second is named *DynamicHistoryBuffer* and has the ability to adapt its size depending on how many entries are needed. Both of these classes follow the same structure, with the only difference being the ability to change in size.

A custom data structure is used to manage the storage of the solution states. This is made up of four components. The main three of these are: a vector which holds the time values for which each solution state corresponds to; a matrix which holds the solution states; and a matrix that holds their respective derivatives (computed simply by evaluating the right-hand-side of the DDE). The initial length of each of these is predetermined by the chosen step size and the maximum delay value present in the system, i.e., how much buffer storage would be required to store states back in time up to and including $t_n - \tau_{\max}$. For the dynamic buffer, this initial length is estimated using a step size of 0.01, as this is the initial step size that is chosen by the adaptive-step solvers to start their stepping. If and when more storage is required, the buffer expands itself automatically by inserting in extra columns. When the buffer is full and the span covers a greater range than is needed (from t_n back to $t_n - \tau_{\max}$), then instead of adding more storage to the buffer, the earliest state is removed and replaced by the new computed state. This is where the fourth component is utilised. Instead of shifting all the states in the buffer to the left so that the earliest is always at the beginning, it is much more computationally efficient to include a single integer variable that stores the index at which the earliest state is stored, thus creating a cyclic storage system. This index is updated as and when the earliest states get overwritten. Any state can be accessed simply by using modular arithmetic with a modulus of the buffer length.

To prevent the numerous passings of states between classes, the buffer classes are designed to also contain the interpolation method required to compute a delay state for a given time point that falls within the range of values in the time buffer. This works by first finding the index in the time buffer of the nearest value to the desired delay time. Checks are made to determine whether the stored value is larger or smaller than the delay time value, thus allowing for the indices of the two neighbouring

states to be found. Due to the cyclic design of the buffer, it is important to remember that neighbouring states may not be directly adjacent to each other in the buffer matrix. Once the pair of column indices are determined, the Hermite interpolation algorithm, given by Eq. (249) in Appendix A.1.2, is employed to compute and return the state at the required time point. If the interpolation method is called with an argument of just the delay time value then it will compute the whole delay state for every variable. There is also an overloaded method that allows for an extra argument to be included to specify distinct indices of the solution vector for which the interpolated solution is required. This is used by the sparse solvers to only compute the delay state values that are required, instead of the entire state.

On top of the methods already detailed, the classes also contain a Boolean check to determine if the buffer is full, along with 'getters' for each individual buffer and the index of the earliest state. The constructors for both classes take in the size of the system and the initial buffer length, with the dynamic buffer also requiring the maximum delay value so that it can adapt the size of the buffer when more storage is required.

4.5.4 *DelayRungeKuttaSolver Classes*

Building upon the *RungeKuttaSolver* class for solving ODEs, we introduce a new class, named *DelayRungeKuttaSolver*, that implements the fixed-step and adaptive-step Runge-Kutta algorithms for solving DDEs. The main stepping algorithm works analogously to its ODE counterpart. Where the delay version differs is the added computation of the delay state matrix, Z, in order to then compute the right-hand-side of the problem in question. The Z matrix is populated by looping through the delays and calling either the `ComputeHistory()` method from the given DDE problem (deriving from *DDEInterface*) or the interpolation algorithm in the history buffer, depending on whether the delay state falls before the initial time, t_0. The

Z matrix has to be generated for every single stage of the Runge-Kutta algorithm, thereby adding a huge amount of computational workload to the time-stepping process, especially when there are a large number of distinct delays in the system.

Thus far, these are the only two delay solvers that have been implemented as part of this package. Similarly to the ODE solvers, the included solvers are *DelayRungeKutta3Solver* and *DelayRungeKutta32Solver*, where the former is a fixed-step 3rd order scheme and the latter is a 3rd order adaptive-step scheme with an embedded 2nd order method for adapting the step size. Due to the Hermite interpolation scheme being of 3rd order, employing a Runge-Kutta stepper of any order higher than this will always result in an overall method of 3rd order [133]. There is potential for lower order methods to be included; however, the computational hardware requirements of such methods will still remain on the same order no matter what the overall order of the scheme. This means that the main trade-off will be between method accuracy and computation time. The 3rd order methods that are provided strike a healthy balance between these factors, while effectively meeting the needs of the intended use. An important point to note is that due to the interpolation algorithm requiring a known solution state either side of each delay time that a solution is required for, this presents the constraint that the step size of the time-stepper may not at any point be larger than the smallest delay in the system. If this were to happen, the latter stages of each Runge-Kutta step could cause some delay times to fall after the most recently computed state, thereby resulting in no known solution state being present ahead of the delay times. This constraint means that for systems with small delays, the step size remains restricted to a small value, thus the system could take a long time to evolve temporally.

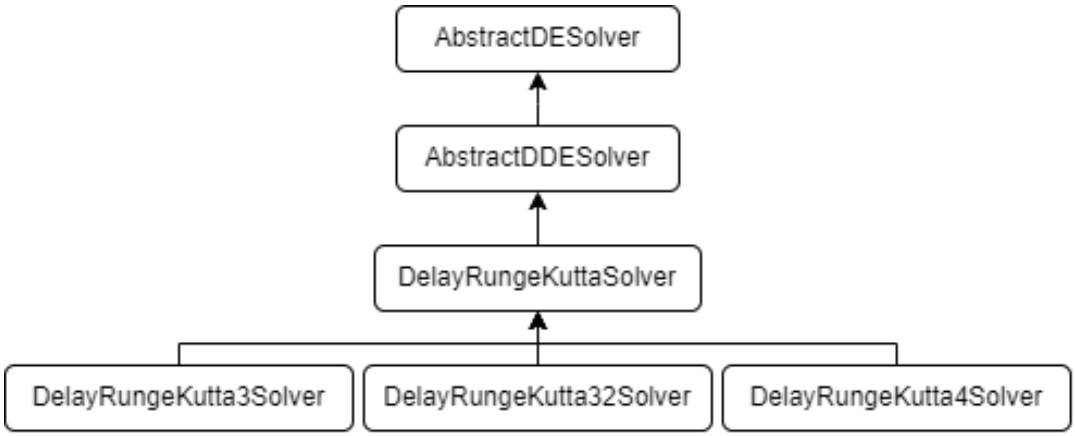


Figure 10: Diagram illustrating the inheritance-based structure of the DDE solver classes.

Above, in Figure 10, we illustrate the class structure of the delay solver classes.

4.6 SPARSE DDE SOLVER DESIGN

The DDE algorithms detailed thus far are similar to those offered by other DDE solver suites. However, for certain problems, there are computational savings that can be made to greatly reduce the number of operations carried out, thus reducing the time of computation for the solution. These savings were first noted when considering the solution of the delayed neural field equation. For this problem, the delays are space-dependent, meaning that the delays between each node in the system are typically dependent on some physical measure of the distance between them. The result of this particular set-up is that when the `ComputeF()` method is called by the solver, not every entry in the Z matrix is used. This raises the question of whether the computation of these unused entries in the Z matrix can be omitted completely.

As an example, let us assume that the delays in a system are defined as $\tau_{i,j} = \phi\left(\left\|\boldsymbol{x}_i - \boldsymbol{x}_j\right\|\right)$, where $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are two spatial points

corresponding to the ith and jth equation in the system, and ϕ is some function of the distance between them (i.e., Euclidean, geodesic, etc.). The algorithm in the standard DDE classes would populate the Z matrix column for a given delay by computing the delayed state of every single state variable at time $t - \tau_{i,j}$. However, if the right-hand-side for the ith equation in the system is only dependent on the delayed state of the jth variable at time $t - \tau_{i,j}$, then the Z matrix column would only need to be populated with the jth entry instead of every single entry. In a NFE, the 'information' passing between two nodes is not dependent on the delay between any nodes other than those two. This allows for any computations of unused values in the Z matrix to be completely ignored.

4.6.1 *SparseDelayMatrix class*

To most efficiently reduce the memory and number of computations required, a sparse matrix data structure can be used to store the Z matrix values. The difference to conventional sparse matrices, however, is that in this case the word 'sparse' does not mean containing few non-zero elements; it instead means containing few 'used' elements. Armadillo has a built in sparse matrix structure, but as it is sparse in the conventional sense it does not allow for zero elements to be stored as values.

As it is the columns of the Z matrix that are looped over when computing the delay states, it makes sense that Z be stored in Compressed Sparse Column (CSC) format as opposed to Compressed Sparse Row (CSR) format, due to the way the elements are stored. More detail on how this storage format works is detailed in Appendix A.3.1.
The constructor for this class takes in a $2 \times m$ matrix containing the index locations, in column-major ordering, of the 'used' delay state matrix values. This can be thought of as the sparsity pattern of the matrix. Here m is the total number of 'used' values. It also takes in an array of length m

containing the values to store in the correspondingly indexed locations, as well as the total number of rows and columns the delay matrix contains. A future consideration, as detailed in the 'Future work' section of this thesis (Section 8.2), would be to determine the sparsity pattern automatically based on an initial 'dry' run of the `ComputeF()` method of the DDE in question that would indicate which entries of the matrix are being used. The class also includes an overloaded parentheses operator, `()`, to access the entries of matrix under conventional matrix index notation, along with methods to return the values from specific individual columns and the corresponding row indices of each of the entries in a column.

4.6.2 *Remaining Sparse Classes*

The remaining sparse classes follow on from analogously their DDE counterparts. Firstly, an interface for all DDE problems backed with the sparse implementation is supplied, under the name *SparseDDEInterface*. This class contains pure virtual methods to compute the right-hand-side and history function of the DDE, utilising the sparse delay matrix structure for Z instead of a standard Armadillo matrix for passing in the delay states to the `ComputeF()` method. Similarly to the previously discussed DDE code, an abstract class to hold all the common properties of DDE solvers which uses the sparse implementation of the delay state matrix is also included. Named *AbstractSparseDDESolver*, this class derives from *AbstractDESolver* and stores a pointer to an object of type *SparseDDEInterface* for computations on the derived specific DDE problem, along with the $2 \times m$ locations matrix used for constructing the sparse delay matrix, Z.

Finally, there is the suite of delay Runge-Kutta solvers that utilise the sparse implementation. A base class, deriving from *AbstractSparseDDE-Solver*, is included to implement the non-adaptive and adaptive generalised Runge-Kutta algorithms. This class is named *DelaySparseRungeKuttaSolver*.

Within the Runge-Kutta implementation methods, the sparse delay matrix, Z, is instantiated using the locations matrix that the user will provide. At each Runge-Kutta step, when the delay times are looped over to populate the columns of Z, the methods in the *SparseDelayMatrix* class are used to return the row indices for the 'used' elements in each column. These row indices are then passed into the `ComputeDelayState()` method of the history buffer to allow it to compute only the relevant entries instead of the entire delay state, thereby greatly reducing the number of computations required.

Deriving further from this class are two specific solver classes. Similarly to the standard DDE solvers, there is a fixed-step 3rd order solver class, named *DelaySparseRungeKutta3Solver*, and an adaptive-step 3rd order solver class which uses an embedded 2nd order scheme for the step size adaptation, named *DelaySparseRungeKutta32Solver*. These are implemented analogously to the standard DDE versions, except the constructors additionally take in the $2 \times m$ locations matrix dictating the sparsity pattern of delay state matrix, Z. These simple adjustments in implementation, compared to other existing DDE solver libraries, have a considerable effect on the performance of the code, allowing for extremely large delayed systems to be solved on lower specification machines with increased efficiency.

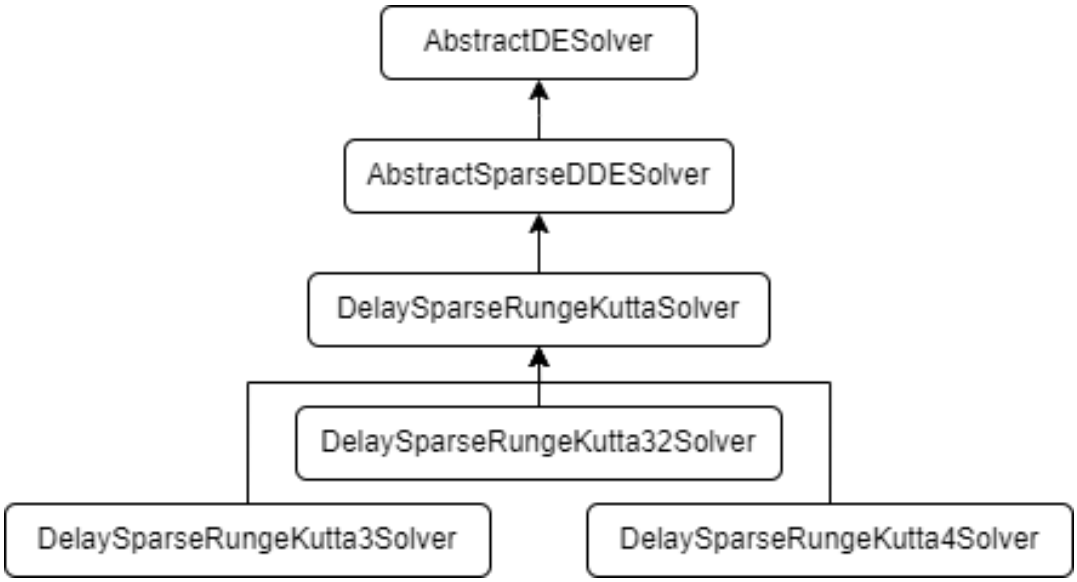


Figure 11: Diagram illustrating the inheritance-based structure of the sparse DDE solver classes.

Figure 11 depicts a class diagram showing the structure that we have just described for the sparse DDE solver classes.

4.7 PARALLELISATION

As mentioned previously, Armadillo has built-in capabilities to handle parallel computations if a relevant opportunity arises, such as computationally expensive element-wise operations. It does this with the use of the OpenMP API specification for parallel programming [43]. The DDE solvers that have been developed thus far (both standard and sparse) all have sections where there is scope to divide up work across multiple processors in order to maximise computational efficiency. As OpenMP is already utilised by Armadillo, it naturally made sense to extend its use to the NFESOLVE library, instead of considering more complicated options such as Message Passing Interface (MPI) [178, 165, 179]. It is also extremely intuitive and easy to implement with existing code.

The main area where speed-up is possible is within the population of the delay state matrix, Z. As this is done column by column, and all the computations are independent of one another, the work can be split up into chunks and distributed amongst the available processors. This is handled automatically by using an OpenMP 'for' loop with shared access to Z.
When the library is compiled, it generates both a sequential and a parallel version of the code. The user then simply has to link to the parallel version and compile with the `-fopenmp` compilation flag if they wish to use the parallel capabilities of the code. Depending on the specific equation being solved, there may also be scope for the user to parallelise their overridden `ComputeF()` method. This is particularly relevant for neural field models as the matrix-vector product of the discretised convolution term can be parallelised. Naturally, it must be noted that when there are only a small number of equations or delays in the system, the overheads generated by parallelising the code may be greater than that of a standard sequential loop, leading to an inefficient execution time.

4.8 LIST OF USER REQUIREMENTS

Now that we have outlined all the classes that make up the NFESOLVE library, we will summarise a list of items that the user must provide in order to utilise the library. The README file from the GitHub repository, which details the installation steps for NFESOLVE, can be found in Appendix B.1.

1. The main thing that the user will need to supply to the solver is a class containing the differential equation that they wish to solve in its discretised form. This class will inherit from either *ODEInterface*, *DDEInterface*, or *SparseDDEInterface*, depending on the problem, and will override the pure virtual `ComputeF()` method found in all of these interface classes in order to specify the discretised right-hand-side of

the equation in question. The user may customise this class however they wish, including adding any additional methods they require. The only requirement is that it inherits from one of the *Interface* classes and overrides the `ComputeF()` method.

2. If the user wishes to make use of the mesh handler toolkit to import a geometric mesh, they may do so by supplying a `.dat` file containing the mesh details in the form illustrated in Section 4.3.1.

3. To make use of the quadrature library to return a set of integration points and weights, the user must supply a mesh via one of the subclasses of *Mesh*. If they wish to use these to solve an integro-differential equation, they must apply them manually inside the class from step 1 containing their discretised differential equation.

A selection of example codes illustrating these steps can be found in the GitHub repository.

4.9 SUMMARY

This chapter gives a full overview into the design, structure, and implementation of a new and bespoke suite of numerical differential equation solvers. We began by discussing the motivation behind why this library is needed and the gaps that are present in existing codebases and solver functions, particularly when it comes to the solution of large-scale delayed systems such as neural field models. The NFESOLVE library aims to eliminate these gaps where possible and make it simple for the user to evolve a variety of differential equations in an efficient and timely manner.

In Section 4.2, we presented the considerations made prior to the commencement of any implementation, detailing the reasoning behind important decisions such as the choice of programming language that the library is built in. The sections that follow this are split into the

individual sub-packages that make up the NFESOLVE library. The first of these sections discusses the design choices and implementation of a mesh handler sub-package to facilitate the storage and manipulation of a geometric mesh. Included as part of this is a quadrature library that allows the user to generate quadrature points and weights on a given mesh. This component is designed particularly with respect to neural field models, which require quadrature rules to numerically approximate the spatial integral terms that are typically present. The next section focuses on the classes that make up the ODE solution suite. Here, we break down the inheritance-based structure that allows the code to be built in a flexible and modular way. This is important as it is applied analogously to the DDE solvers that are discussed in the following two sections. Following the structural design, we detail the Runge-Kutta solvers that are implemented based on the algorithms explained in Appendix A.1. Section 4.5 then explores the standard DDE solver design, which follows an algorithmic methodology which most other DDE solvers that are currently used in practice also follow, such as MATLAB's dde23 solver. The design of the DDE solver suite is structured very similarly to the ODE sub-package of the NFESOLVE library. After the discussion of the standard DDE solvers, we then proceed to discuss the classes that make up the sparse DDE solver implementation. This begins by introducing a new custom data structure that is built to store the matrix of delay states computed at each Runge-Kutta stage. Rather than a dense matrix (as used in the standard DDE implementation), this class is analogous to a sparse matrix structure, based on the commonly used CSC format. Using this data structure allows for a great reduction in both the number of computations and the amount of storage required to facilitate the temporal evolution of a DDE. The remaining sparse classes that make up the sparse DDE solver sub-package are then introduced. These follow the same inheritance-based design structure that both the ODE and standard DDE implementations follow. Finally, we discuss the opportunity for the parallelisation of the

code and the technologies that are used to accomplish this, along with a summary of the requirements that a user must provide in order to utilise the NFESOLVE library.

The following chapter (Chapter 5) explores several examples and applications used to validate that the code performs as expected. This is discussed from both a numerical convergence analysis perspective, and from a replication of pre-existing neural modelling results perspective. Following on from this, the full capabilities of the NFESOLVE library are employed in Chapters 6 and 7 to evolve models of neural activity in both a discrete neural mass setting and a continuum neural field setting, with the incorporation of real brain data. The NFESOLVE codebase is available to download in full at `https://github.com/UoN-Math-Neuro/NFESOLVE`.

5

VALIDATION OF CODE

5.1 INTRODUCTION

The NFESOLVE library, discussed in Chapter 4, is an innovative and newly developed suite of codes. As with all novel products, it is extremely important that rigorous testing is undertaken to validate that it performs to the quoted specifications. The main intended use of the library is to solve delay differential equations, specifically with a bias towards solving large-scale models of neural dynamics. To show that the code performs as expected, this chapter presents a number of validation methods and examples in a variety of settings.

Firstly, and most importantly, the convergence rates of the schemes are tested to indicate that they behave as expected when the step size or tolerances are decreased. This is done by numerically solving arbitrary problems for which analytic solutions exist, and then comparing the results to the exact solution by means of an error measure. Following on from this, we explore several settings of the neural field equation (NFE), with and without delays, for which exact analytical results are known. This is especially relevant as it confirms if the code yields the expected output when different parameters of the NFE are varied to give different solutions.

5.2 ERROR ANALYSIS

To show that the developed solvers converge with the rates depicted in their respective names (as introduced in Chapter 4), this section features error analyses for a variety of different problems, both with and without delays.

5.2.1 *ODE Solvers*

To start, we consider the solvers implemented in the ODE module of the library. As described in Section 4.4, the non-adaptive solvers developed thus far are the 3rd and 4th order Runge-Kutta solvers, named *RungeKutta3Solver* and *RungeKutta4Solver*, respectively. For the adaptive solvers, currently there is only the 3rd order adaptive Runge-Kutta solver, named *RungeKutta32Solver*, implemented in the library. In order to test the convergence properties of each solver, we use the ODE problem

$$\begin{aligned} y_1'(t) &= y_3(t), & y_1(0) &= 1, \\ y_2'(t) &= -y_3(t), & y_2(0) &= -1, \\ y_3'(t) &= \frac{1}{2}(y_2(t) - y_1(t)), & y_3(0) &= 0. \end{aligned} \tag{87}$$

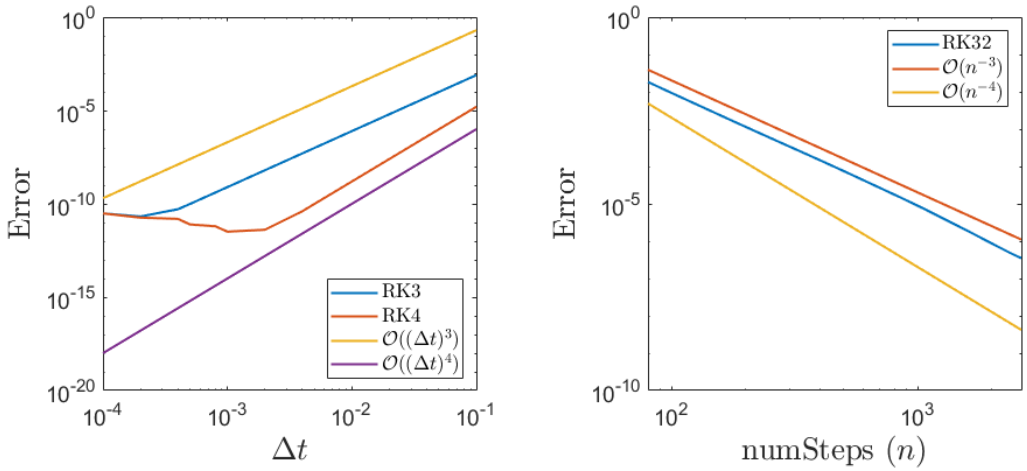
This is an arbitrarily selected ODE for which the analytic solution is known. For the specified initial conditions, this has the trigonometric solutions

$$\begin{aligned} y_1(t) &= \cos(t), \\ y_2(t) &= -\cos(t), \\ y_3(t) &= -\sin(t). \end{aligned} \tag{88}$$

Denoting the true solution in vector form by $\boldsymbol{y}$, and the numerical approximation of the solution by $\widehat{\boldsymbol{y}}$, we use the superscript index s to denote the solution at the sth temporal point, i.e., $\boldsymbol{y}^s = \boldsymbol{y}(t^s)$. Taking an error measure of the infinity norm, defined by

$$\|\boldsymbol{y} - \widehat{\boldsymbol{y}}\|_\infty \approx \max_{i=1,\dots,m} \max_{s=1,\dots,n} |y_i^s - \widehat{y}_i^s|, \tag{89}$$

where m is the number of equations in the system and n is the total number of temporal points from the initial time up to a specific final time, we repeat a number of simulations with different step sizes and tolerances in order to assess the rate of convergence of a given solver. Figure 12 depicts the convergence rates of each ODE solver in the NFESOLVE library, for the problem defined above by Eq. (87). For the non-adaptive solvers (Figure 12a), the error is plotted against the fixed time step, Δt. As the adaptive solver (Figure 12b) is controlled by varying the absolute and relative error tolerances that make up the adaptation condition, to quantitatively show the convergence we plot the error against the total number of steps the method takes to reach the final time, t^n. For this example, a final time of $t^n = 20$ was chosen.



(a) Non-adaptive ODE solvers.

(b) Adaptive ODE solver.

Figure 12: Convergence plots depicting the rate at which the numerical solution of Eq. (87) converges to the true solution Eq. (88), using both non-adaptive and adaptive ODE solvers from the NFESOLVE package.

To illustrate the order of convergence for each solver, arbitrary 3rd and 4th order lines are plotted alongside the error. Note that the convergence plots tail off when the error reaches $\approx 10^{-}11$, as this is where machine precision errors pollute any computed values. It is clear that the solvers perform to

their expected order. This is a key result in the validation of the code as it indicates that the ODE suite is implemented correctly.

5.2.2 *DDE Solvers*

Similarly to the ODE solvers, we perform a convergence analysis in an analogous way for the DDE solver suite. As the sparse DDE solvers use the exact same time-stepping and interpolation algorithms as the standard DDE solvers, we present this analysis for just the standard solvers, namely *DelayRungeKutta3Solver* and *DelayRungeKutta32Solver*. The sparse solvers contribute towards improving computation speed and memory requirements, which is particularly relevant only when considering larger systems. As discussed in Appendix A.2.3 and Section 4.5.4, the overall order of a DDE solver is defined by the minimum of the order of the time-stepping scheme and the order of the interpolation method, therefore, it is expected that both the non-adaptive and adaptive solvers will be 3rd order convergent [133].

Consider the DDE problem

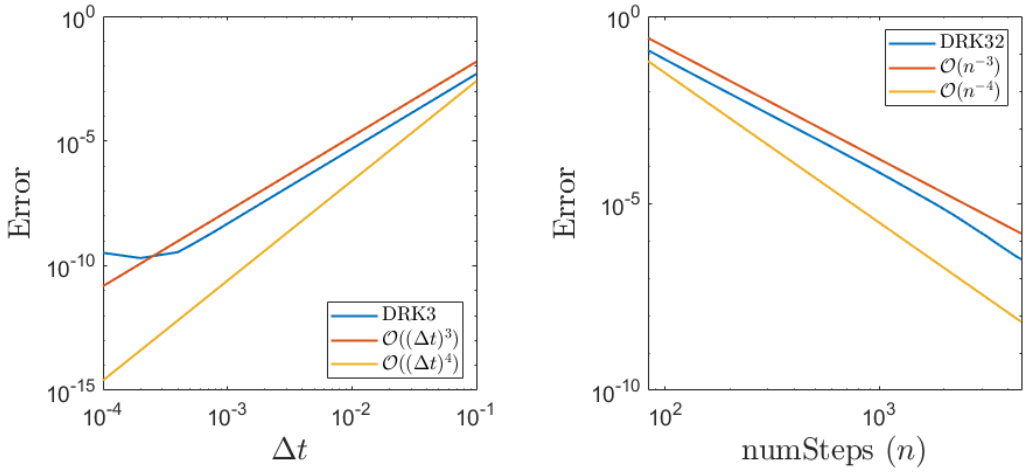
$$\begin{aligned} y_1'(t) &= -y_1\left(t - \frac{\pi}{2}\right), \\ y_2'(t) &= y_3(t), \\ y_3'(t) &= \left(y_1(t-\pi)\right)^2 - y_1\left(t - \frac{\pi}{4}\right) - y_2(t). \end{aligned} \tag{90}$$

with the history function for y_1 given by $\varphi_1(t) = \cos(t) + \sin(t)$, for $t < 0$, and the initial conditions $y_2(0) = y_3(0) = 0$. Although it is rare to find

analytic solutions to DDEs, this problem was specifically engineered such that it gives the solution

$$\begin{aligned} y_1(t) &= \cos(t) + \sin(t), \\ y_2(t) &= 1 + \left(\frac{\sqrt{2}}{2}t - 1\right)\cos(t) + \left(\frac{2}{3} - \frac{\sqrt{2}}{2}\right)\sin(t) - \frac{1}{3}\sin(2t), \\ y_3(t) &= \frac{2}{3}(\cos(t) - \cos(2t)) + \left(1 - \frac{\sqrt{2}}{2}t\right)\sin(t). \end{aligned} \tag{91}$$

Using the same error norm as in the previous section, defined by Eq. (89), we repeat simulations for a variety of step sizes and error tolerances to construct the convergence plots depicted in Figure 13. Once again, the error of the adaptive solver is plotted against the total number of steps taken to reach the final time, t^n, which in this case is taken to be $t^n = 20$.



(a) Non-adaptive DDE solver.

(b) Adaptive DDE solver.

Figure 13: Convergence plots depicting the rate at which the numerical solution of Eq. (90) converges to the true solution Eq. (91), using both non-adaptive and adaptive DDE solvers from the NFESOLVE library.

Likewise to Figure 12, both plots in Figure 13 also contain arbitrary 3rd and 4th order lines to clearly demonstrate the rate at which the solver error decreases. Evidently, both the non-adaptive and adaptive DDE solvers per-

form to 3rd order as expected, thereby validating that the implementation of the DDE suite is correct.

5.3 NEURAL FIELD EQUATION ON A SPHERE

There has been a vast amount of research undertaken with NFEs posed on one and two dimensional domains ([38, 64, 117, 59] to name just a few); however, when it comes to applying them to surfaces in three dimensions considerably less exploration has been done to date. Ideally, a realistic cortical mesh will be the domain of choice when solving NFEs due to the primary reasoning behind them being to model real-life brain dynamics. A very naive 'brain' can be modelled by using a spherical geometry. This also allows for some interesting analysis to be undertaken, as where analytic solutions exist, they can be broken down into linear combinations of basis functions known as spherical harmonics. In this section, we look at some results and pattern formations for a neural field posed on a spherical domain, based on work by Visser et al. [177]. Following this, we consider the dynamics of the NFE when a Heaviside firing rate is employed and a solution in the shape of a spherical cap is enforced. This is useful in validating the code in the NFESOLVE library as there are theoretical results that the numerical solutions can be compared against.

5.3.1 *Spherical Harmonics*

Before discussing the model and its dynamics, it is important to first introduce the spherical harmonics as they are an integral part to the analysis undertaken. The spherical harmonics are a set of complex-valued functions that form an orthonormal basis on sphere; analogous to Fourier series on a circle. They were first introduced in 1782 by Laplace (see [118] for a history of spherical harmonics) and are formulated from the angular portion of the

solution to the Laplace equation posed on a sphere. We use the notation $Y_n^m(\boldsymbol{r})$ to detail an nth degree, mth order spherical harmonic, where $n \geq 0$ and $|m| \leq n$, with $\boldsymbol{r} = (r, \theta, \phi)$ being a point on a sphere with radius, r, polar angle, $\theta = [0, \pi,]$, azimuthal angle, $\phi = [0, 2\pi)$. One such representation on the unit sphere ($r = 1$) is defined by the functions

$$Y_n^m(\theta, \phi) = (-1)^m \sqrt{\frac{2n+1}{4\pi} \frac{(n-m)!}{(n+m)!}} P_n^m(\cos\theta)\, e^{im\phi}, \quad -n \leq m \leq n, \tag{92}$$

where P_n^m is the associated Legendre function [4] given by

$$P_n^m(x) = \frac{\left(1-x^2\right)^{m/2}}{2^n n!} \frac{d^{n+m}}{dx^{n+m}} \left(x^2-1\right)^n. \tag{93}$$

An important result is that the harmonics satisfy the equation

$$Y_n^{-m}(\theta, \phi) = (-1)^m \overline{Y_n^m(\theta, \phi)}, \tag{94}$$

where $\overline{\ }$ represents complex conjugation. They also obey the orthogonality condition

$$\int_0^{2\pi} \int_0^{\pi} Y_n^m(\theta, \phi)\, \overline{Y_{n'}^{m'}(\theta, \phi)} \sin\theta \, d\theta \, d\phi = \delta_{n,n'} \delta_{m,m'}. \tag{95}$$

A selection of example spherical harmonics are depicted below in Figure 14.

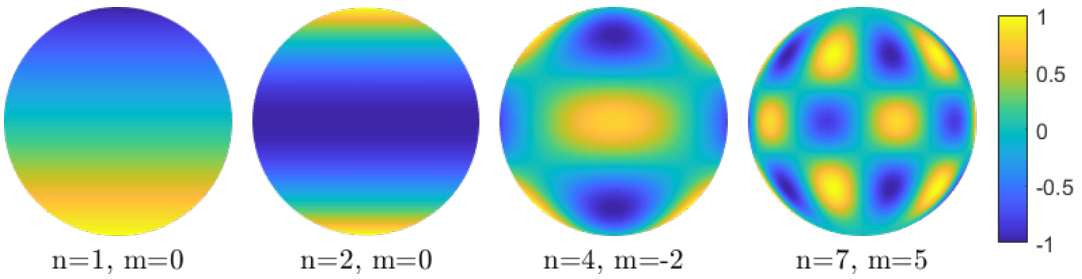


Figure 14: Examples of the real part of four different spherical harmonics for (n, m) pairs: $(1, 0), (2, 0), (4, -2), (7, 5)$. Each has been normalised to the domain $[-1, 1]$ for visualisation purposes.

As the spherical harmonics are complex functions, for illustration purposes we have plotted just the real part for each (n, m) pair, and normalised each solution to the domain $[-1, 1]$.

5.3.2 *The Model*

Let us consider a scalar neural field equation of the form

$$\frac{\partial}{\partial t}u\left(\boldsymbol{r},t\right)=-u\left(\boldsymbol{r},t\right)+\int_{\Omega}w\left(\boldsymbol{r},\boldsymbol{r}'\right)f\left(u\left(\boldsymbol{r}',t\right)\right)d\boldsymbol{r}', \tag{96}$$

where the domain $\Omega := S^2$ is the surface of the unit sphere in $\mathbb{R}^3$, w is a connectivity kernel facilitating the non-local connections, f is the firing rate function, and the state variable, u, represents activity.. Here, $\boldsymbol{r}=(\theta,\phi)$ is a point on the sphere with polar angle, $\theta = [0,\pi,]$, azimuthal angle, $\phi = [0,2\pi)$ and radius 1. On a sphere, the geodesic distance between any two points is the distance along the great circle that the points share. For the unit sphere, this is equal to the angular separation of the points, given by the inverse cosine of the dot product of the two points. Hence, we select the connectivity kernel, w, such that it is a function of the dot product of the two points, i.e., $w\left(\boldsymbol{r},\boldsymbol{r}'\right)=w\left(\boldsymbol{r}\cdot\boldsymbol{r}'\right)$.

5.3.3 *Linear Stability Analysis*

Firstly, we make note of some key functional analysis results, presented by Visser et al. [177], that we use in this section. Given a function, w, posed on $[-1,1]$, the surface integral, $I=\int_{\Omega}w\left(\boldsymbol{r}\cdot\boldsymbol{r}'\right)\mathrm{d}\boldsymbol{r}'$, can be transformed via the change of variables, $s=\cos\left(\theta\right)=\boldsymbol{r}\cdot\boldsymbol{r}'$, into an integral of a single variable, $I=2\pi\int_{-1}^{1}w\left(s\right)\mathrm{d}s$. Secondly, the function, w, can be written as an infinite sum of the spherical harmonics, i.e.

$$w\left(s\right)=w\left(\boldsymbol{r}\cdot\boldsymbol{r}'\right)=\sum_{n=0}^{\infty}w_n\sum_{m=-n}^{n}Y_n^m\left(\boldsymbol{r}\right)\overline{Y_n^m\left(\boldsymbol{r}'\right)}, \tag{97}$$

where the coefficients w_n are given by

$$w_n=2\pi\int_{-1}^{1}w\left(s\right)P_n\left(s\right)\mathrm{d}s, \tag{98}$$

where P_n is the Legendre polynomial of degree n. Furthermore, we say that w is *balanced* if

$$w_0=2\pi\int_{-1}^{1}w\left(s\right)\mathrm{d}s=0. \tag{99}$$

It will become clear where these results are used to aid in the linear stability analysis of the model.

For this section, we consider the homogeneous steady states of the system given by

$$\hat{u} = f(\hat{u})\, w_0. \tag{100}$$

If the connectivity kernel is balanced (i.e. Eq. (99) is satisfied), then the only homogeneous steady state of the system that satisfies Eq. (100) is the zero state, $\hat{u} = 0$. Assuming a solution of the form $u(\boldsymbol{r}, t) = \hat{u} + v(\boldsymbol{r}, t)$, where v is some small perturbation of the form $v(\boldsymbol{r}, t) = \mathrm{e}^{\lambda t} q(\boldsymbol{r})$, for $\lambda \in \mathbb{C}$, we substitute this into equation Eq. (96) and linearise to give the equation

$$(\lambda + 1)\, q(\boldsymbol{r}) = f'(\hat{u}) \int_\Omega w\left(\boldsymbol{r} \cdot \boldsymbol{r}'\right) q\left(\boldsymbol{r}'\right) \mathrm{d}\boldsymbol{r}'. \tag{101}$$

From here on, let us use the substitution, $\gamma = f'(\hat{u})$, for readability. Considering solutions of the form $q(\boldsymbol{r}) = Y_{n'}^{m'}(\boldsymbol{r})$ and expanding w as a sum of spherical harmonics (making use of Eq. (97)), the linearised equation, Eq. (101), can be written as

$$(\lambda + 1)\, Y_{n'}^{m'}(\boldsymbol{r}) = \gamma \sum_{n=0}^{\infty} w_n \sum_{m=-n}^{n} Y_n^m(\boldsymbol{r}) \int_\Omega Y_{n'}^{m'}\left(\boldsymbol{r}'\right) \overline{Y_n^m(\boldsymbol{r}')}\, \mathrm{d}\boldsymbol{r}' \tag{102}$$

$$= \gamma \sum_{n=0}^{\infty} w_n \sum_{m=-n}^{n} Y_n^m(\boldsymbol{r})\, \delta_{n,n'} \delta_{m,m'}, \tag{103}$$

where w_n are as given in Eq. (98). There only exists non-trivial solutions to this equation in the case that $n = n'$ *and* $m = m'$, hence we formulate the eigenvalue equation

$$\mathcal{E}(\lambda) := \lambda + 1 - \gamma w_n = 0, \qquad n \geq 0. \tag{104}$$

There exist $2n + 1$ eigenfunctions of the system, following the form $v(\boldsymbol{r}, t) = \mathrm{e}^{\lambda t} Y_n^m(\boldsymbol{r})$, for $m = -n, \ldots, n$.

5.3.4 *Choice of Kernel*

A common choice of kernel is the sum of exponentials given by the function

$$w(s) = J_1 \exp\left(-\frac{\cos^{-1}(s)}{\sigma_1}\right) + J_2 \exp\left(-\frac{\cos^{-1}(s)}{\sigma_2}\right), \tag{105}$$

where $J_1 J_2 < 0$ and $\sigma_2 > \sigma_1 > 0$. To give the kernel Mexican hat shape, we also enforce the condition $J_1 + J_2 > 0$. In order to determine the stability of the system with this kernel, it is necessary to compute the coeffecients, w_n. Visser et al. [177] show that for an integral of the form

$$I_n(a) = \int_{-1}^{1} \exp\left(a \cos^{-1}(s)\right) P_n(s)\, \mathrm{d}s, \tag{106}$$

where $a \in \mathbb{C}$ and P_n is the Legendre polynomial of degree $n \geq 0$, the recursive relation:

$$\begin{aligned} I_n(a)\, I_{n+1}(a) &= \frac{a^2\left(1 - \mathrm{e}^{2a\pi}\right)}{\left(a^2 + n^2\right)\left(a^2 + (n+1)^2\right)\left(a^2 + (n+2)^2\right)}, \\ I_0(a) &= \frac{1 + \mathrm{e}^{a\pi}}{a^2 + 1}, \end{aligned} \tag{107}$$

is satisfied. This also yields the two-term recursive relation:

$$I_{n+2}(a) = I_n(a)\, \frac{a^2 + n^2}{a^2 + (n+3)^2}. \tag{108}$$

Using this, the coefficients w_n are easily constructed, allowing the solutions to the eigenvalue equation (Eq. (104)) to be generated.

For ease, we fix the parameters $J_2 = -1$ and $\sigma_2 = 1$ in Eq. (105). Using the balance condition, Eq. (99), an expression for the parameter J_1 can be formulated in terms of σ_1 to always enforce a balanced kernel:

$$J_1 = \frac{\mathrm{e}^{-\pi}\left(1 + \mathrm{e}^{\pi}\right)\left(1 + \sigma_1^2\right)}{2\left(1 + \mathrm{e}^{-\frac{\pi}{\sigma_1}}\right)\sigma_1^2}. \tag{109}$$

Taking all of this into account, the kernel has now been reduced from being in terms of four parameters down to only a single parameter $\sigma = \sigma_1$.

Figure 15 shows an example of this reduced connectivity kernel with a parameter value of $\sigma = 0.4$.

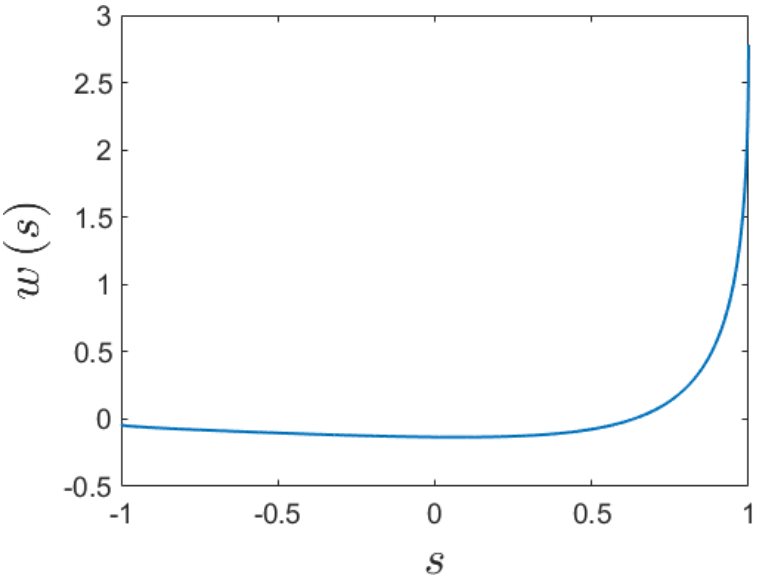


Figure 15: Balanced connectivity kernel with $\sigma = 0.4$.

Using the recursion formulae (Eq. (107)) detailed above, the coefficients w_n of the kernel are calculated as

$$w_n = 2\pi \left(\frac{e^{-\pi} \left(1 + e^{\pi}\right) \left(1 + \sigma^2\right)}{2 \left(1 + e^{-\frac{\pi}{\sigma}}\right) \sigma^2} I_n \left(\frac{-1}{\sigma} \right) - I_n \left(-1\right) \right). \tag{110}$$

This expression allows for the eigenvalue spectra to be computed for all $n \geq 0$.

5.3.5 *Modal Instabilities*

As the eigenvalue equation given by Eq. (104) only supports real solutions, for each mode we expect only stationary patterned solution states to arise from instabilities of the homogeneous steady state. Let us first consider the sigmoidal firing rate function

$$f\left(u\right) = \frac{1}{1 + e^{-\mu(u-\vartheta)}}, \tag{111}$$

where the parameter μ controls the steepness of the sigmoid and ϑ is the threshold parameter. Starting from the zero steady state, $\hat{u} = 0$, if all n

eigenvalues of the system are negative then the state remains stable and does not change. To illustrate the modal instabilities present in the system, we look in turn at the instabilities that arise when only the nth eigenvalue is pushed to be unstable but all others remain in the stable region. To simplify the search for a parameter regime for which the different individual modes become unstable, we fix the firing rate steepness parameter, μ, to have a value of $\mu = 30$. This leaves two parameters: the firing rate threshold, ϑ, and the connectivity kernel parameter , σ. Adjusting these to allow only one eigenvalue to go unstable at a time, the eigenvalue spectra depicted in Figure 16 show example instabilities for $n = 1, \ldots, 6$. The specific values of the parameters for which the depicted instabilities occur can be seen below in Table 1.

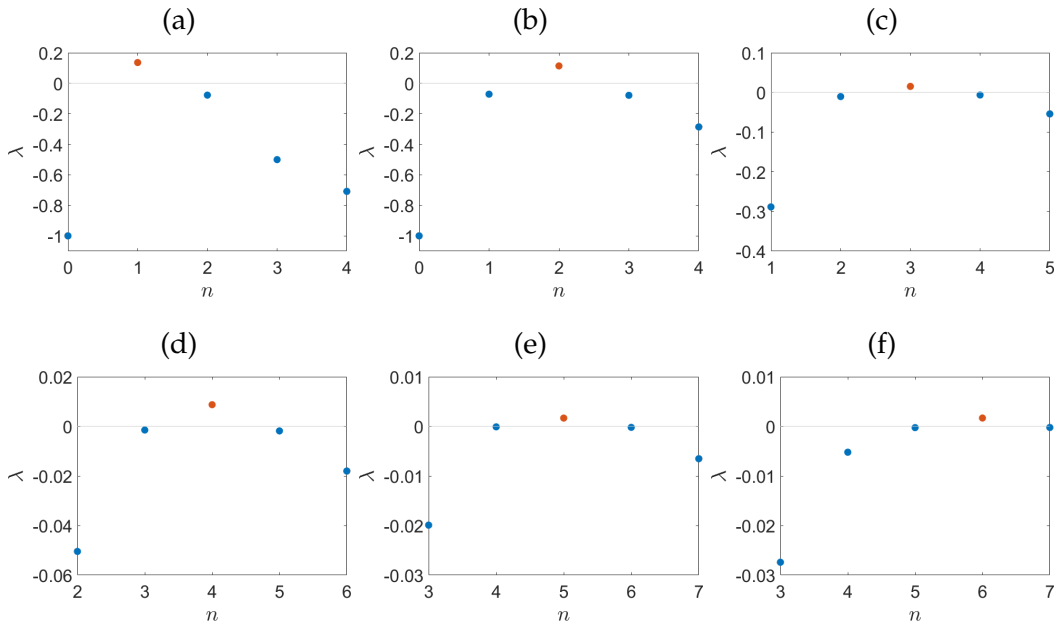


Figure 16: Eigenvalue spectra showing modal instabilities for $n = 1, \ldots, 6$ (corresponding to (a)-(f)). Here, the red dots represent the unstable modes and the blue dots show the stable modes. The specific parameter values for each plot can be found in Table 1.

n	σ	ϑ
1	0.5	0.1
2	0.2	0.133
3	0.065	0.1476
4	0.0355	0.15
5	0.02	0.15118
6	0.0135	0.15158

Table 1: Parameter values used for the modal instabilities depicted in Figure 16 and Figure 17.

As n increases, the spherical harmonics that make up the eigenfunctions become more complicated. This is reflected in the numerical solutions shown in Figure 17, which show the instabilities corresponding to the spectra and parameter values above in Figure 16 and Table 1. Due to the Runge-Kutta algorithm behind the non-adaptive and adaptive solvers being the same, all numerical solutions in this section were computed using the adaptive-step *RungeKutta32Solver* ODE solver from the NFESOLVE library in order to minimise computation time.

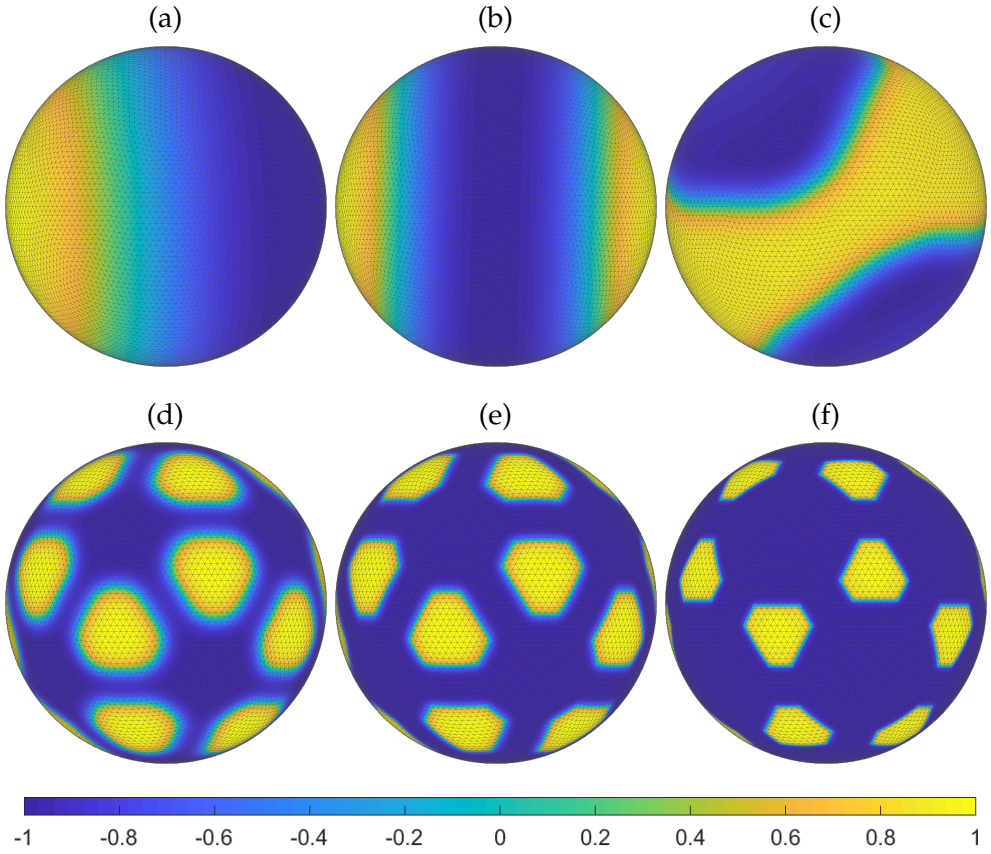


Figure 17: Instabilities of the zero state, $u(\boldsymbol{r}) = 0$, on a triangulated mesh of the unit sphere made up of 10242 nodes, computed using the NFESOLVE library's *RungeKutta32Solver* ODE solver. Figures (a)-(f) show the instabilities of the nth mode for $n = 1, \ldots, 6$. The yellow areas represent high 'activity' and dark blue represents low 'activity', with each plot being normalised to the domain $[-1, 1]$ for visualisation purposes.

The solution states start off as a simple gradiented pattern from one hemisphere to the other, before gradually becoming more complicated and forming more intricate patterns. If certain groups of modes are pushed unstable at the same time then the patterns that arise are linearly combined to achieve intricate solutions. In the presence of delays, as shown in Visser et al. [177], Hopf bifurcations naturally occur, thereby giving rise to oscillating and travelling wave solutions across the surface of the sphere. As this

validation example was undertaken during the development phase of the DDE code, this section only considers the non-delay version of the model. It is clear that the NFESOLVE library performs to expectations and yields solutions to the model predicted by the theory.

5.3.6 *Spherical Cap*

Let us now consider the model in a similar setting but with a Heaviside firing rate function instead of a sigmoid, i.e.

$$f(u) = H(u - \vartheta) = \begin{cases} 1, & u \geq \vartheta, \\ 0, & u < \vartheta. \end{cases} \tag{112}$$

For this validation example, we wish to look for solutions in the form of a '*cap*' on the unit sphere. In spherical coordinates, this will be defined in terms of the polar angle, $\theta \in [0, \pi]$. Under the assumption that the solution is azimuthally homogeneous, i.e., $u(\boldsymbol{r}, t) \to q(\theta)$, the steady state solution will be such that

$$\begin{aligned} &q(\theta_c) = \vartheta, \\ &q(\theta) > \vartheta, \qquad \text{for } \theta < \theta_c, \\ &q(\theta) < \vartheta, \qquad \text{for } \theta > \theta_c, \end{aligned} \tag{113}$$

for some $\theta_c \in [0, \pi]$. The radius Δ of the cap is given by $\Delta = \sin\theta_c$. The steady state analysis of Eq. (96) leads to the solution

$$q(\theta) = \int_0^{2\pi} \int_0^{\theta_c} w\left(\boldsymbol{r} \cdot \boldsymbol{r}'\right) \sin\theta' \, \mathrm{d}\theta' \, \mathrm{d}\phi'. \tag{114}$$

In spherical coordinates, $\boldsymbol{r} = (x, y, z) = (\sin\theta\cos\phi, \sin\theta\sin\phi, \cos\theta)$, hence, the dot product $\boldsymbol{r} \cdot \boldsymbol{r}'$ can be written as

$$\boldsymbol{r} \cdot \boldsymbol{r}' = \sin\theta\cos\phi\sin\theta'\cos\phi' + \sin\theta\sin\phi\sin\theta'\sin\phi' + \cos\theta\cos\theta' \tag{115}$$

$$= \sin\theta\sin\theta'\cos\left(\phi' - \phi\right) + \cos\theta\cos\theta'. \tag{116}$$

Shifting $\phi' \to \phi' - \phi$ and using the invariance of the integral with respect to ϕ' over $[0, 2\pi]$, we reach the expression

$$q(\theta) = \int_0^{2\pi} \int_0^{\theta_c} w \left(\sin\theta \sin\theta' \cos\phi' + \cos\theta \cos\theta' \right) \sin\theta' \, \mathrm{d}\theta' \, \mathrm{d}\phi'. \tag{117}$$

This gives an implicit equation for θ_c in the form

$$q(\theta_c) = \vartheta, \tag{118}$$

which can be used to numerically solve the equation for a chosen value of θ_c, thereby determining the threshold value, ϑ, necessary to give a cap solution of radius $\Delta = \sin\theta_c$.

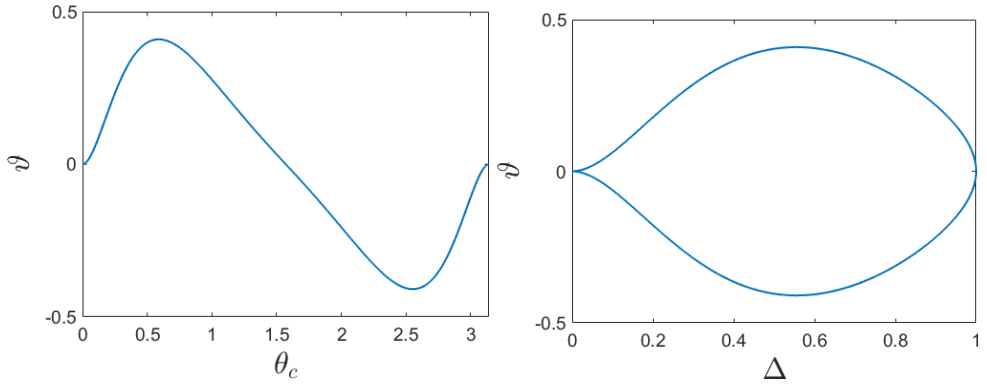


Figure 18: Numerically computed plots of the firing rate threshold, ϑ, against $\theta_c \in [0, \pi]$ (left), and for ϑ against the cap radius, Δ (right). The chosen connectivity kernel, w, is that depicted in Figure 15 (with parameter $\sigma = 0.4$).

Let us consider an example problem that assumes the balanced connectivity kernel shown in Figure 15 (with parameter $\sigma = 0.4$). Numerically solving Eq. (118) for the full range of θ_c allows for the construction of the plots, shown in Figure 18, of ϑ vs. θ_c and ϑ vs. Δ. Figure 19 shows the spherical cap solutions for two different values of θ_c. Initial data was chosen pertaining to the condition given in Eq. (113) and the simulations were run until a steady state solution was reached.

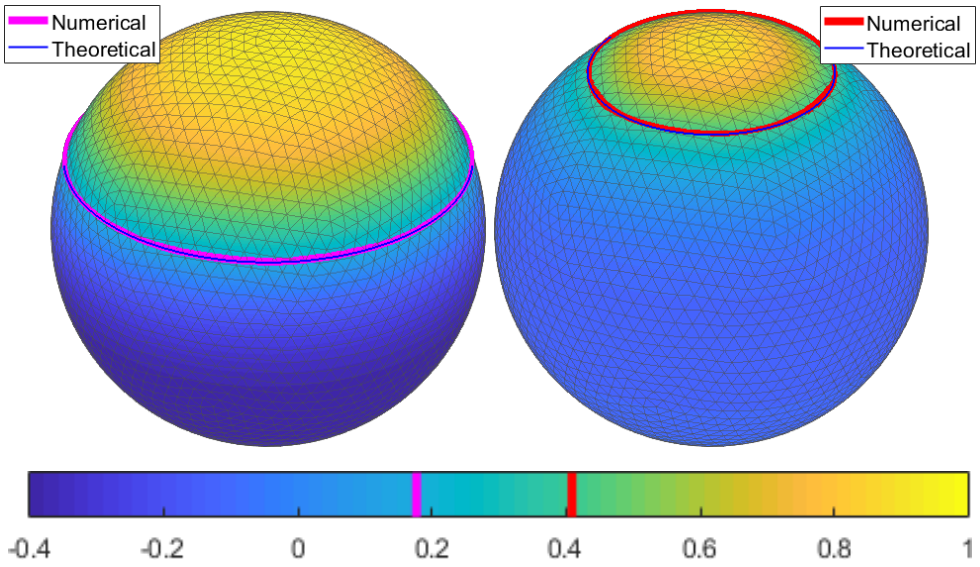


Figure 19: Spherical cap solution on a 2562 node triangulated unit sphere mesh, for $\theta_c = 1.2$ (left) and $\theta_c = 0.6$ (right), therefore producing a cap of radius $\Delta \approx 0.932$ and $\Delta \approx 0.565$, respectively. The corresponding firing rate threshold values used for these simulations are $\vartheta \approx 0.177$ and $\vartheta \approx 0.409$. The numerical boundary of the caps are illustrated with the magenta and red circles, and the theoretical boundary (computed using Eq. (118)) are illustrated by blue circles.

As expected, the cap solution $q(\theta)$ is equal to ϑ (illustrated with magenta and red circles) at the points illustrated with blue circles, which show the theoretical cap boundary.

Both the examples in this section clearly show that the NFESOLVE library can be confidently used to compute numerical solutions to NFEs. This is illustrated by the fact that the numerical solutions conform to the expected theoretical results for the problems considered.

5.4 NEURAL FIELD EQUATION WITH A SINGLE CONSTANT DELAY

Although the full spatially-dependent delay NFE introduced is more physically relevant for describing cortical activity, we first wish to explore the dynamics of the NFE with just a single constant delay that applies to the whole network. In [142, 143, 125], Roxin et al. discuss the significance of fixed delays in neuronal rate models. It was found that just the presence of a fixed delay was enough to introduce waves and oscillations into a system which would not otherwise support such solutions. These dynamic solutions typically arise via instabilities of steady states; however, their existence depends heavily on the connectivity kernel and firing rate function chosen within the model. The one-dimensional NFE with a single constant delay is given by

$$\frac{\partial}{\partial t}u\left(x,t\right)=-u\left(x,t\right)+\int_{\mathbb{R}}w\left(x,y\right)f\left(u\left(y,t-\tau\right)\right)\mathrm{d}y, \tag{119}$$

where $x\in\mathbb{R}$. Once again, the firing rate function, f, is the taken to be the sigmoidal function given by Eq. (111), with parameters μ and θ dictating the steepness and threshold of the sigmoid, respectively. We also impose the condition that the connectivity kernel, w, is defined such that $w\left(x,y\right)=w\left(x-y\right)=w\left(\left|x-y\right|\right)$, i.e., that it is translationally invariant.

5.4.1 *Linear Stability Analysis*

Firstly, by exploiting the translational invariance of the kernel, the integral term in Eq. (119) can be re-written to give

$$\frac{\partial}{\partial t}u\left(x,t\right)=-u\left(x,t\right)+\int_{\mathbb{R}}w\left(y\right)f\left(u\left(x-y,t-\tau\right)\right)\mathrm{d}y. \tag{120}$$

The steady state of this equation is given by the solution to

$$\bar{u}\left(x\right)=\int_{\mathbb{R}}w\left(y\right)f\left(\bar{u}\left(x-y\right)\right)\mathrm{d}y. \tag{121}$$

Considering only homogeneous steady states, i.e., $\overline{u}(x) = \overline{u}$ for all x, this allows the steady state equation to be written as

$$\overline{u} = f(\overline{u}) \int_{\mathbb{R}} w(y)\, \mathrm{d}y. \tag{122}$$

A key technique that can be used in the stability analysis of equations in this form is the Fourier transform. The one-dimensional Fourier transform of w, as introduced and defined in Section 2.3.4, is given by

$$\widetilde{w}(k) = \int_{-\infty}^{\infty} w(y)\, \mathrm{e}^{iky}\, \mathrm{d}y. \tag{123}$$

Employing this, the steady state equation for homogeneous steady states can be written succinctly as

$$\overline{u} = f(\overline{u})\, \widetilde{w}(0). \tag{124}$$

Let $u(x,t) = \overline{u} + v(x,t)$ for some small perturbation of the form $v(x,t) = \mathrm{e}^{\lambda t}\, \mathrm{e}^{ikx}$, where $\lambda \in \mathbb{C}$ and $k \in \mathbb{R}$. We substitute this into Eq. (119) and linearise to yield the transcendental eigenvalue equation

$$\mathcal{E}(\lambda, k) := \lambda + 1 - \gamma \widetilde{w}(k)\, \mathrm{e}^{-\lambda\tau} = 0, \tag{125}$$

where $\gamma = f'(\overline{u})$. Depending on the values of λ and k, there are four different types of instabilities that are possible. For $\mathrm{Re}(\lambda) > 0$:

- Steady: $\mathrm{Im}(\lambda) = 0$ and $k = 0$. This leads to a global uniform change in activity.
- Turing: $\mathrm{Im}(\lambda) \neq 0$ and $k = 0$. This leads to the formation of an inhomogeneous stationary state.
- Hopf: $\mathrm{Im}(\lambda) = 0$ and $k \neq 0$. This leads to global spatially periodic oscillations of a uniform state.
- Turing-Hopf: $\mathrm{Im}(\lambda) \neq 0$ and $k \neq 0$. This leads to travelling wave solutions.

Let $\lambda = \nu + i\omega$, where $\nu, \omega \in \mathbb{R}$. Splitting Eq. (125) into its real and imaginary parts yields the two equations

$$\text{Re:} \qquad \nu + 1 - \gamma \widetilde{w}(k)\, \mathrm{e}^{-\nu\tau} \cos(\omega\tau) = 0, \tag{126}$$

$$\text{Im:} \qquad \omega + \gamma \widetilde{w}(k)\, \mathrm{e}^{-\nu\tau} \sin(\omega\tau) = 0. \tag{127}$$

By dividing these two equations, we deduce that

$$\tan(\omega\tau) = -\frac{\omega}{1+\nu}. \tag{128}$$

Considering the point of bifurcation ($\nu = 0$), we see that a Hopf or Turing-Hopf bifurcation arises for the non-zero solution to the transcendental equation $\tan(\omega\tau) = -\omega$. It is clear from Figure 20 that the first non-zero solution lies in the range $\pi/2\tau < \omega < \pi/\tau$. As the period of oscillation, T, of the resulting bifurcation is given by $T = 2\pi/\omega$, this means that $2\tau < T < 4\tau$.

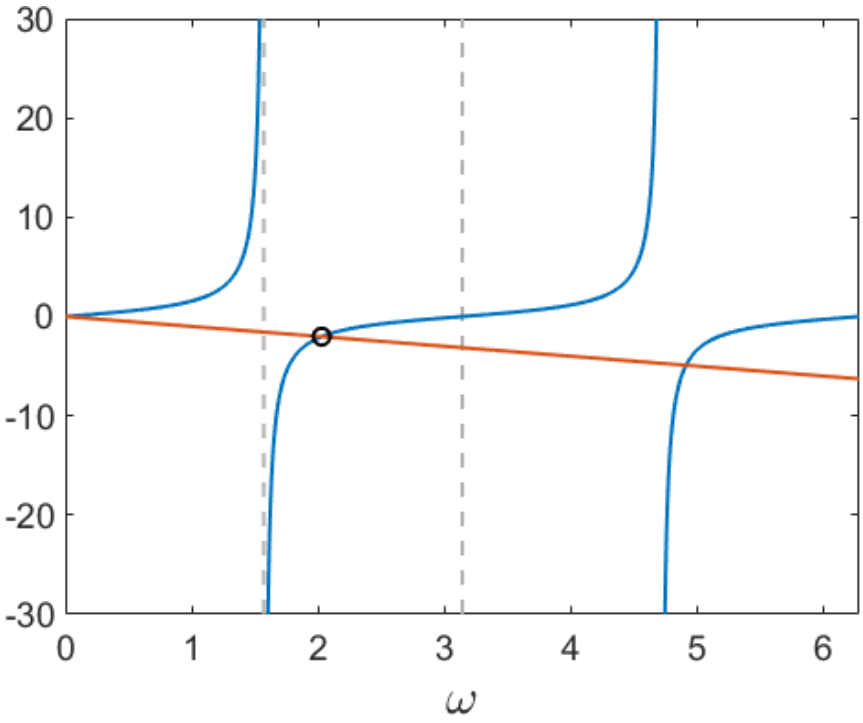


Figure 20: Plot of $\tan(\omega\tau)$ (blue) and $-\omega$ (red), for $\tau = 1$, to show first non-zero intersection point (black). Dotted grey lines illustrate $\omega = \pi/2\tau$ and $\omega = \pi/\tau$.

Rearranging Eq. (126) yields

$$\cos(\omega\tau) = \frac{1}{\gamma\tilde{w}(k)}, \tag{129}$$

and this is only satisfied when $\gamma\tilde{w}(k) < -1$, as $\cos(\omega\tau)$ is negative for $\pi/2\tau < \omega < \pi/\tau$. As the firing rate function, f, is taken to be an increasing sigmoid, the derivative of it at any point is always positive, hence $\tilde{w}(k)$

must be less than 0 for a Hopf or Turing-Hopf bifurcation to exist. Taking Eq. (125), this may be rearrange to yield

$$(\lambda + 1)\, \mathrm{e}^{\lambda\tau} = \gamma \widetilde{w}\,(k)\,. \tag{130}$$

Multiplying both sides by $\tau\, \mathrm{e}^{\tau}$, the equation becomes

$$\tau\,(\lambda + 1)\, \mathrm{e}^{\tau(\lambda+1)} = \gamma \widetilde{w}\,(k)\, \tau\, \mathrm{e}^{\tau}, \tag{131}$$

which we can then write in terms of the Lambert W function as

$$\tau\,(\lambda + 1) = W\left(\gamma \widetilde{w}\,(k)\, \tau\, \mathrm{e}^{\tau}\right). \tag{132}$$

The Lambert W function [41] (also known as the product logarithm) is defined as the multivalued function that satisfies $z = W(z)\, \mathrm{e}^{W(z)}$, for any complex number, z. In this case, we use the extension that for the equation, $z = xa^{x}$, the solution is given by the identity, $x = {}^{W(z \ln a)}/_{\ln a}$. Rearranging for λ, we reach the final solution

$$\lambda\,(k) = -1 + \frac{1}{\tau} W_n\left(\gamma \widetilde{w}\,(k)\, \tau\, \mathrm{e}^{\tau}\right), \tag{133}$$

for $n \in \mathbb{Z}$, where W_n is the nth branch of the Lambert W function. In the absence of delay, i.e., taking the limit as $\tau \to 0^{+}$, the solution to the eigenvalue problem is $\lambda\,(k) = -1 + \gamma \widetilde{w}\,(k)$, as expected.

5.4.2 *Simulation Results*

Let us consider the inverted Mexican hat kernel, $w\,(x) = (-1 + |x|)\, \mathrm{e}^{-|x|}$. This type of connectivity kernel depicts local inhibition and lateral excitation, which when looking at large-scale brain dynamics is a natural choice of kernel [34]. The Fourier transform of $w\,(x)$ is

$$\widetilde{w}\,(k) = \int_{-\infty}^{\infty} (-1 + |y|)\, \mathrm{e}^{-|y|}\, \mathrm{e}^{iky}\, \mathrm{d}y = -\frac{4k^2}{\left(1 + k^2\right)^2}, \tag{134}$$

which is less than or equal to zero for all k.

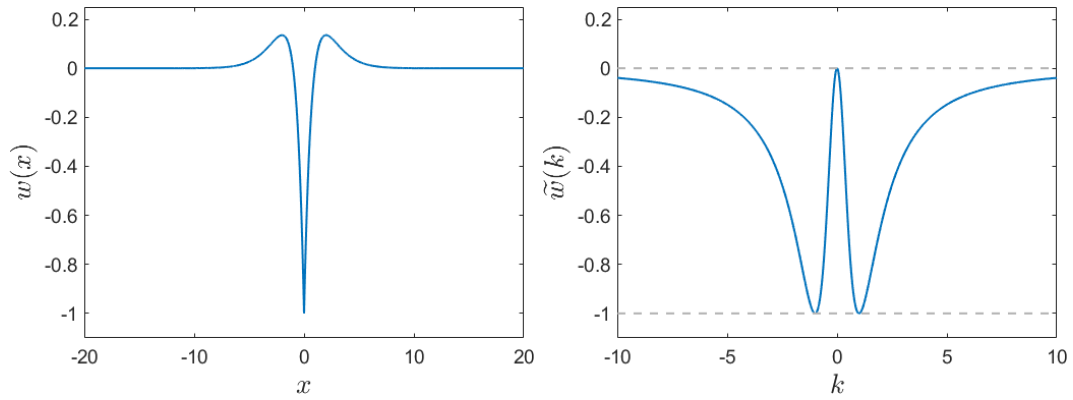


Figure 21: Plot of the inverted Mexican hat kerne,l $w(x) = (-1 + |x|)\, e^{-|x|}$, (left) and its Fourier transform, $\widetilde{w}(k) = -4k^2/(1+k^2)^2$, (right).

This has local minima at $k = \pm 1$ with a value of -1, hence $k_c = \pm 1$ are the emergent wave numbers when excited from below. As the kernel is balanced ($\widetilde{w}(0) = 0$), the steady state we look to excite is the zero state $\overline{u} = 0$. The eigenvalue equation Eq. (125) for $k = 0$ gives $\lambda = -1$ and therefore a Hopf instability cannot occur. However, as $\widetilde{w}(k_c)$ is negative, it satisfies the condition Eq. (129) for a Turing-Hopf bifurcation to exist. From the condition $\gamma\widetilde{w}(k_c) < -1$, we see that γ must therefore be greater than 1. For fixed values of τ, we numerically solve $\tan(\omega\tau) = -\omega$ for the first non-zero value of ω. For each of these, we may then compute γ via Eq. (129), yielding $\gamma = -1/\cos(\omega\tau)$. This allows for the construction of a two-parameter bifurcation diagram of the parameters γ and τ (Figure 22). Depicted are the stability regions separated by the Turing-Hopf bifurcation. In region I, the homogeneous steady state remains stable; however, in region II, perturbations to the steady state yield dynamic periodic travelling wave solutions.

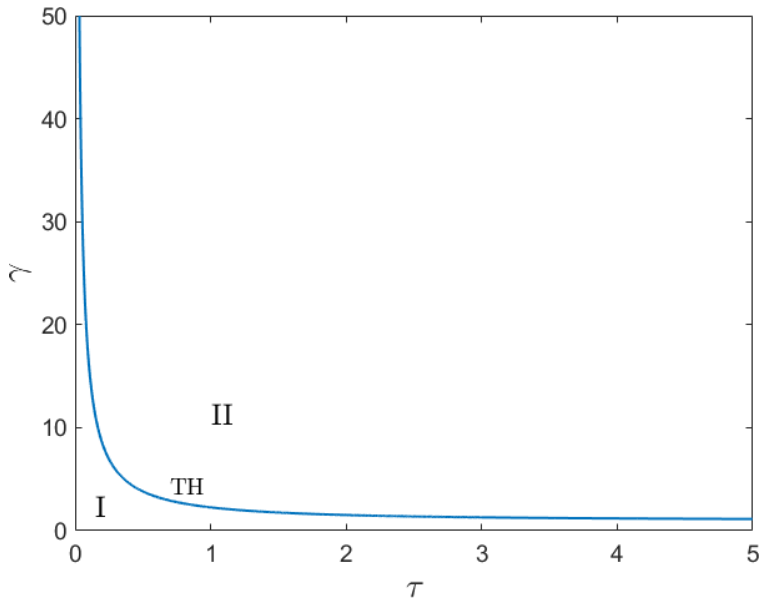


Figure 22: Two-parameter bifurcation diagram for γ against τ. The homogeneous steady state $\overline{u}$ is stable in I. The blue line depicts a Turing-Hopf bifurcation (TH). In II, the steady state is subject to a Turing-Hopf instability and gives rise to a periodic travelling wave.

It is evident that as τ tends to zero from above, the first non-zero solution to $\tan(\omega\tau) = -\omega$ tends to the lower solution bound (illustrated in Figure 20) of $\pi/2\tau$, hence γ tends to infinity at the point of bifurcation. However, for the case where there is explicitly no delay present in the system (i.e., $\tau = 0$), the solution to Eq. (125) requires γ to equal -1 for the eigenvalue to cross the imaginary axis. This is not possible, due to γ being strictly positive for the chosen firing rate function. Hence, the homogeneous steady state remains stable at all times in the absence of delays, under the current model assumptions.

Although it is simple here to consider the gradient of the firing rate function as a parameter itself, γ is actually dependent on the firing rate steepness parameter, μ, and threshold parameter, θ.

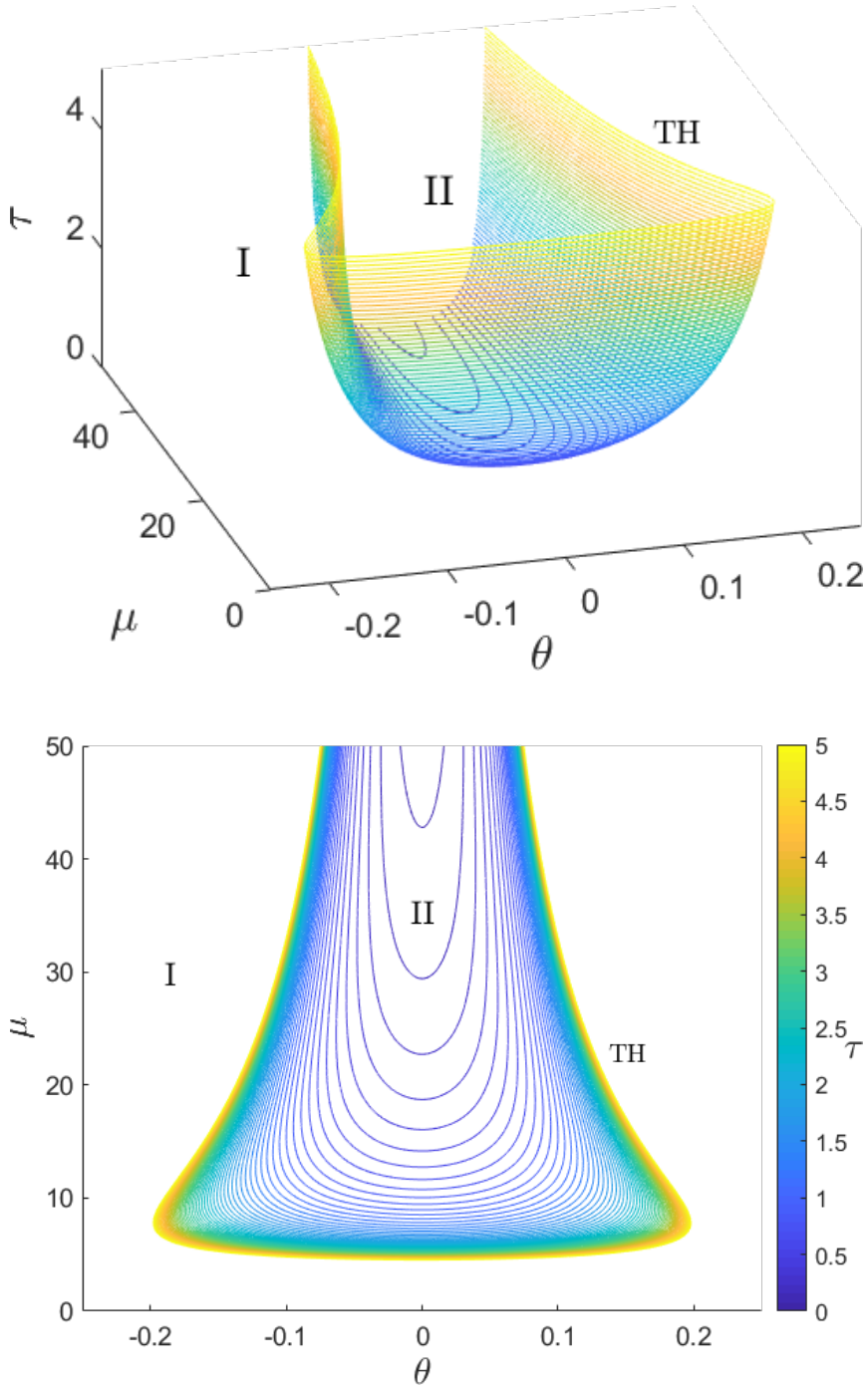


Figure 23: Two views of a three-parameter bifurcation diagram for μ, θ and τ. The regions, I and II, are synonymous with that of Figure 22, and TH illustrates a Turing-Hopf bifurcation.

As these are the parameters required for simulation, it makes sense to further break down the bifurcation condition in terms of these parameters instead of just a single gradient parameter. Numerically solving the eigen-

value problem for a range of μ and θ, we may construct a three-parameter bifurcation diagram for μ, θ, and τ. This is illustrated in Figure 23. The stability regions depicted in this diagram are analogous to the stability regions shown in the two-parameter diagram, with the Turing-Hopf bifurcation now represented by the contours that make up a surface. An example solution, with parameters located in regime II, can be seen below in Figure 24. This solution was computed using the adaptive-step *DelayRungeKutta32Solver* solver from the NFESOLVE library. As there is only a single delay present in the system, it does not make sense to use NFESOLVE's sparse delay solvers, which only yield a computational efficiency boost when working with larger systems with multiple delays. The chosen parameters for this simulation are $\mu = 20$, $\theta = 0.13$ and $\tau = 4$, which do not fall too far from the bifurcation point. As expected, a Turing-Hopf instability arises, with periodic waves travelling towards the centre of the domain from either side. At the point of bifurcation, the period, T, between the waves can be computed as $T = 2\pi/\omega$.

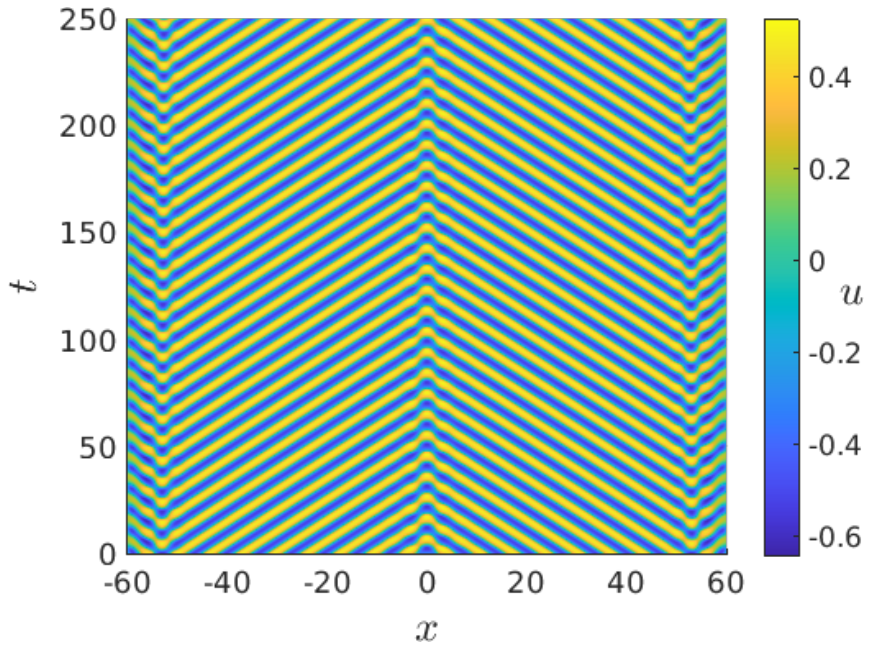


Figure 24: Example periodic travelling wave solution with period, $T \approx 9.733$, emerging from a Turing-Hopf instability. Period was computed by taking the average of the differences between consecutive peaks of the solution. The parameters chosen fall in regime II of Figure 23, with values $\mu = 20$, $\theta = 0.13$, and $\tau = 4$.

For this example, $\omega \approx 0.644$, thereby leading to an expected period of $T \approx 9.753$. This respects the prediction that $2\tau < T < 4\tau$. The period of the numerically computed solution, computed by taking the average of the differences between consecutive peaks of the solution, is approximately equal to 9.733, which conforms closely to the theoretical expectation. As the solution yielded meets the anticipated results, this gives confidence that the NFESOLVE library's delay solvers perform accurately to the expected standard.

For steeper firing rate functions (larger μ), other dynamics such as stationary bumps and global oscillatory patterns can be observed. This is discussed in more detail in [142, 143, 125].

5.5 NEURAL FIELD EQUATION WITH SPACE-DEPENDENT DELAYS

Following on from the single constant delay NFE presented in the previous section, we now explore some results for NFEs with multiple space-dependent delays. This work fully utilises the NFESOLVE suite's capabilities to solve large-scale delay equations, and allows for existing theoretical and numerical results from the literature to be employed in validating that the solvers yield the expected results. As the model contains a large number of delays, for which not all elements of the delay state vectors are required, it makes sense to employ the sparse delay solvers that are provided by the NFESOLVE library. Explanations of how the sparse delay solvers work can be found in Section 4.6. Before exploring any solutions, it is important to first understand how the sparse delay solvers can be set up to maximise computational efficiency. This section then considers bump solutions with transient oscillations, based on work by Faye and Faugeras [61], and subsequently looks at travelling front solutions, based on work by Coombes et al. [37, 35]. Both of these solution types arise due to the presence of axonal delays. This validation step is particularly important as the models considered in Chapters 6 and 7 all incorporate axonal delays.

5.5.1 *Sparse Solver Setup*

Depending on the problem under consideration, using a standard DDE solver can result in a lot unnecessary computations that are ultimately never actually used in any further calculations. This can lead to much higher computation times and memory utilisation. As detailed in Section 4.6, the sparse solvers in the NFESOLVE library are designed to mitigate these consequences by computing only the delay states that are actually required for evaluating the right-hand-side of the DDE. Due to the specific structure of NFEs with space-dependent delays, there are factors that can be exploited

to optimise the set up and sparsity pattern of the delay state matrix, Z, that the sparse solvers require. To illustrate this, we consider the discretised form of the delay NFE [117]. For a full description of this discretisation, see Section 3.3.4.

$$\dot{u}_i(t) = -u_i(t) + \sum_j^n w_{ij} f\left(u_j\left(t - \tau_{ij}\right)\right) \sigma_j, \quad i = 1, \ldots, n. \tag{135}$$

As explained in Section 2.3.2, the delay terms, τ_{ij}, are typically chosen to take the form $\tau_{ij} = \tau_c + \|x_i - x_j\|/v$, for $i, j = 1, \ldots, n$, where τ_c is some constant delay term that applies to all connections. At first glance, for τ_c non-zero, this system contains n^2 delay values as there is a separate delay for each ij pair. However, due the fact that the delays are symmetric ($\tau_{ij} = \tau_{ji}$ for $i, j = 1, \ldots, n$) and also that there is no spatial component to the delay when $j = i$ ($\tau_{ii} = \tau_c$ for $i = 1, \ldots, n$), this allows for the total number of distinct delays to be dramatically cut down from n^2 to $n(n-1)/2 + 1$.

The DDE solvers in the NFESOLVE library require a vector of delays to be passed in when instantiating a solver object. Although the delays do not require a specific ordering, it makes sense to order them in such a way that the ij indices can effortlessly be mapped to the index at which the delay sits in this vector. Firstly, consider the delays in matrix form, where the rows and columns of the matrix are labelled by i and j, respectively, i.e.,

$$\begin{array}{c} \\ i \end{array} \begin{array}{c} \begin{array}{cccc} \multicolumn{4}{c}{j} \end{array} \\ \begin{pmatrix} \tau_{11} & \tau_{12} & \cdots & \tau_{1n} \\ \tau_{21} & \tau_{22} & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \tau_{n1} & \cdots & \cdots & \tau_{nn} \end{pmatrix} \end{array}. \tag{136}$$

As the delays are symmetric, it is only necessary to store the delays with indices $j \geq i$, yielding an upper triangular matrix. On top of this, as the main diagonal entries are all equal, it is only necessary to store the first

entry, τ_{11}. This leaves an upper triangular matrix with a single entry on the main diagonal.

$$\begin{pmatrix} \tau_{11} & \tau_{12} & \tau_{13} & \cdots & \tau_{1n} \\ & & \tau_{23} & \cdots & \vdots \\ & & & \ddots & \vdots \\ & \mathbf{0} & & & \tau_{n-1,n} \\ & & & & \end{pmatrix} \tag{137}$$

Flattening this matrix out row by row, a vector of delays is produced. Denoting the indices of this vector by k, where $k = 1, \ldots, {}^{n(n-1)}/_{2} + 1$, a mapping between the ij indices and the vector index is given by

$$k = \begin{cases} 1, & i = j, \\ n(i-1) - \frac{i(i+1)}{2} + j + 1, & i < j. \end{cases} \tag{138}$$

When looping through to evaluate the right-hand-side of Eq. (135), if a delay is required for indices $j < i$, then the indices may simply be flipped to access the correct delay.

At each stage of the time-stepping process, the delay state matrix, Z, is computed. Each column of this matrix holds the state vector for each delay time. This matrix is of size $n \times ({}^{n(n-1)}/_{2} + 1)$, and therefore contains ${}^{n(n^2-n+2)}/_{2}$ entries. However, due to the form of the discretised delay NFE (Eq. (135)), only n^2 of the entries in the Z matrix are actually required by the solver. This is because for a given delay state, $u_a(t - \tau_{bc})$, for $a, b, c = 1, \ldots, n$, the delay state will only actually be used in the computation of the right-hand-side of Eq. (135) when $c = a$. Therefore, savings can be made by only computing the entries that are needed, rather

than computing the whole delay state. We illustrate this for the case when $n = 4$. The Z matrix will have the form

$$\begin{array}{c} \\ u_1 \\ u_2 \\ u_3 \\ u_4 \end{array} \begin{array}{c} \begin{array}{cccccccc} \tau_{11} & \tau_{12} & \tau_{13} & \tau_{14} & \tau_{23} & \tau_{24} & \tau_{34} \end{array} \\ \begin{pmatrix} F_1 & F_2 & F_3 & F_4 & & & \\ F_2 & F_1 & & & F_3 & F_4 & \\ F_3 & & F_1 & & F_2 & & F_4, \\ F_4 & & & F_1 & & F_2 & F_3 \end{pmatrix} \end{array}, \tag{139}$$

where F_i, $i = 1, \ldots, 4$, represents a computation required in the evaluation of the right hand side of Eq. (135) for the ith equation. For the first delay, every state is required as this is the case when $j = i$. However, for every other delay value, only two states are required. It is clear from this example that, due to the exploitation of the delay symmetry, for a given delay, τ_{ij}, where $i < j$, the only states required for that delay are those corresponding to each index, i.e., u_i and u_j. In the case where there is no constant delay, i.e., $\tau_c = 0$, the first column of the delay state matrix may simply be removed. The total number of delay state values in the Z matrix that are used for computations then becomes $n(n-1)$, and the mapping between the ij indices and the delay vector index, k, is given by

$$k = n\,(i-1) - \frac{i\,(i+1)}{2} + j. \tag{140}$$

For either case, it is simple to construct the sparsity pattern required by the sparse delay solvers.

This setup is the most computationally efficient way we have found to utilise the sparse delay solvers in the NFESOLVE library to solve delay NFEs with space-dependent delays. It drastically cuts down the number of computations required to populate the Z matrix at each stage of the Runge-Kutta algorithm from $n\,(n^2 - n + 2)/2$ to n^2, in the case of $\tau_c \neq 0$, which is a considerable difference when working with large systems as this is a whole order of magnitude smaller. We utilise this setup for solving all space-dependent delay NFEs in this thesis.

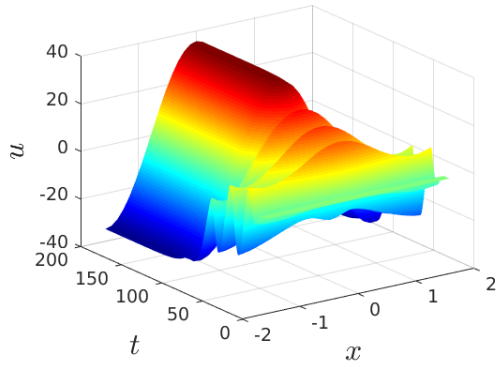
5.5.2 *Bump Solutions*

In [61], Faye and Faugeras consider a network of neurons that mimics the architecture of a cortical hypercolumn in the primary visual cortex. Instead of thinking of the neurons as having a spatial component x, they instead treat the neurons as being labelled by an angle, θ, which ranges from $-\pi/2$ to $\pi/2$, for orientation preference in visual processing. This model was originally conceived by Ben-Yishai et al. [18]. The mean activity response, u, in the network (with space-dependent delays) is given by the equation

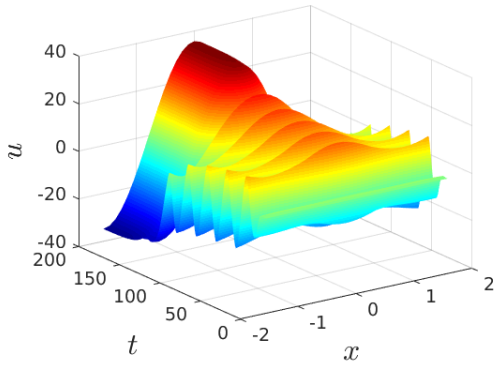
$$\begin{aligned}\tau\frac{\partial}{\partial t}u(\theta,t) = &-u(\theta,t)+\int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} w(\theta-\theta')\, f\left(u\left(\theta',t-\frac{|\theta-\theta'|}{v}\right)\right)\frac{\mathrm{d}\theta'}{\pi}\\ &+\epsilon I(\theta-\theta_0)-\theta_{\text{th}}.\end{aligned} \tag{141}$$

Here, the connectivity kernel, w, is selected to be sinusoidal function, $w(\theta) = w_0 + w_1\cos(2\theta)$, and the nonlinearity, f, is the sigmoidal function, $f(u) = (1+\mathrm{e}^{-\mu u})^{-1} - 1/2$. Also included in the model is an external input term, I, chosen to take the form $I(\theta) = 1-\beta+\beta\cos(2\theta)$. This is incorporated along with a scaling parameter, ϵ. Other parameters in this model are the temporal scaling parameter, τ, the orientation for which the external input is maximal, θ_0, and the neuronal threshold, θ_{th}. Although the model is describing a slightly different phenomenon than seen in previous examples, it is fundamentally in the same form as the delay NFEs that we wish to explore in the upcoming chapters.

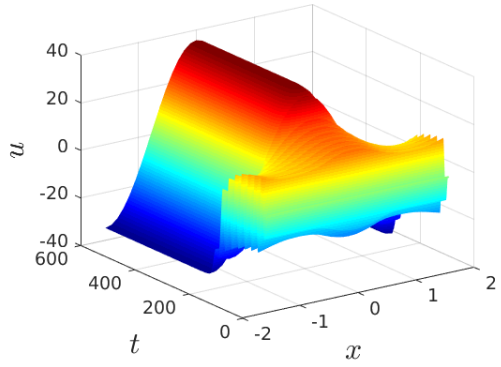
We have recreated the results of Faye and Faugeras in order to validate that the NFESOLVE solvers generate the same results presented in [61]. Typical parameters values used in [18], that we fix across all simulations, are $\beta = 0.1$, $w_0 = -73$, $w_1 = 110$, $\epsilon = 1.45$, $\tau = 10$, $\theta_0 = 0$ and $\theta_{\text{th}} = 1$. Faye and Faugeras state that they were unable to locate in the literature a suitable biological value for the axonal delay speed, v, and therefore select it have a value of $v = 0.2\text{rad}\,\text{s}^{-1}$.



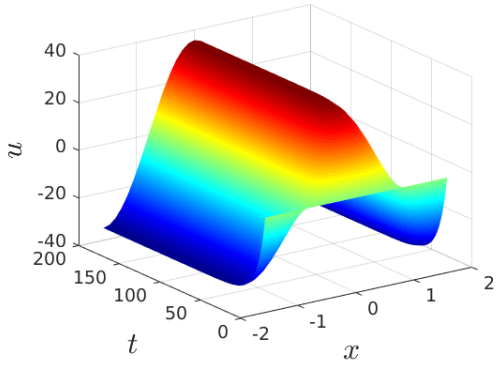
(a) Solution of Eq. (141) for $\mu = 3$.



(b) Solution of Eq. (141) for $\mu = 13$.



(c) Solution of Eq. (141) for $\mu = 100$.



(d) Solution of Eq. (141) with no delays.

Figure 25: Solutions of Eq. (141) with varying sigmoid slopes. Parameter values used in all simulations are: $\beta = 0.1$, $w_0 = -73$, $w_1 = 110$, $\epsilon = 1.45$, $\tau = 10$, $\theta_0 = 0$ and $\theta_{\text{th}} = 1$.

Figure 25 depicts the difference in solution for varying values of the sigmoid steepness parameter, μ. Using the sparsity pattern detailed in Section 5.5.1, we employ the adaptive-step *SparseDelayRungeKutta32Solver* solver from the NFESOLVE library to generate each solution. An initial condition close to $u(x, 0) = 0$ is used for all of these simulations. As μ increases, it is evident that the transient oscillations appear to last for longer before settling down into a 'bump' solution. For comparison, the solution to the non-delay problem is also included. These results conform to the expected output seen in Faye and Faugeras' work, thereby validating that the sparse delay solver implementation is correct. Their paper proceeds to

discuss results for two populations of neurons in one dimension, and also a selection of results in two dimensions. We do not consider these results here, however, as this work is purely a validation exercise based on the standard delayed neural field model.

5.5.3 *Travelling Front Solutions*

Coombes et al. present an analysis of the one-dimensional delayed NFE in [37, 35], where they also derive the Evans function for the stability of wave solutions. We aim to employ some of the results from their paper in order to generate travelling front solutions for which the theoretical front speed is known. This may then be used to compare against the travelling front speed of the numerical solution to verify that the selected solver yields expected results.

In the case of travelling wave solutions, it is common to shift to the comoving frame by using the change of variables $\xi = x - ct$. We wish to seek functions, $u(x,t) = q(\xi)$, which satisfy the NFE (Eq. (96)) with space-dependent delays. This results in the travelling front equation for $q(\xi)$:

$$-c\frac{\mathrm{d}}{\mathrm{d}\xi}q(\xi) = -q(\xi) + \int_{-\infty}^{\infty} w(y)\, f\left(q\left(\xi - y + \frac{c|y|}{v}\right)\right)\mathrm{d}y. \tag{142}$$

Considering the firing rate function, f, to be the Heaviside function, denoted by $f(u) = H(u-\theta)$, we look for front solutions such that $q(\xi) > \theta$ for $\xi < 0$, and $q(\xi) < \theta$ for $\xi > 0$. Note that it physically does not make sense for the wave speed, c, to be greater than the axonal delay speed, v, and so we enforce the restriction that $c < v$. The integral part of Eq. (142) becomes

$$\int_{-\infty}^{\infty} w(y)\, H\left(q\left(\xi - y + \frac{c|y|}{v}\right) - \theta\right)\mathrm{d}y := \psi(\xi). \tag{143}$$

For the assumed form of q, this integral can be simplified to remove the Heaviside term. When $q(\xi - y + {}^{c|y|}/_{v})$ is greater than θ, the evaluated

Heaviside function is equal to 1, otherwise it is 0. Therefore, the following condition arises for the integrand to remain non-zero:

$$\xi - y + \frac{c\,|y|}{v} < 0. \tag{144}$$

It follows from this that there are two cases to consider: i) that for $\xi \geq 0$, and ii) that for $\xi < 0$. For the first of these cases, Eq. (144) is valid for $y > \xi/(1 - c/v)$. For case ii), Eq. (144) is valid for $y > \xi/(1 + c/v)$. Hence, the integral can be rewritten as

$$\psi(\xi) = \begin{cases} \int_{\frac{\xi}{1-c/v}}^{\infty} w(y)\,\mathrm{d}y, & \xi \geq 0, \\ \int_{\frac{\xi}{1+c/v}}^{\infty} w(y)\,\mathrm{d}y, & \xi < 0. \end{cases} \tag{145}$$

We may then solve

$$-c\frac{\mathrm{d}}{\mathrm{d}\xi}q(\xi) = -q(\xi) + \psi(\xi), \tag{146}$$

using an integrating factor, giving

$$\frac{\mathrm{d}}{\mathrm{d}\xi}\left[q(\xi)\,\mathrm{e}^{-\xi/c}\right] = -\frac{1}{c}\psi(\xi)\,\mathrm{e}^{-\xi/c}. \tag{147}$$

Fixing an origin with the choice $q(0) = \theta$ allows Eq. (147) to be integrated as

$$\int_0^{\xi} \frac{\mathrm{d}}{\mathrm{d}\xi'}\left[q(\xi')\,\mathrm{e}^{-\xi'/c}\right]\mathrm{d}\xi' = -\int_0^{\xi} \frac{1}{c}\psi(\xi')\,\mathrm{e}^{-\xi'/c}\,\mathrm{d}\xi'. \tag{148}$$

This then simplifies to give

$$q(\xi) = \mathrm{e}^{\xi/c}\left[\theta - \frac{1}{c}\int_0^{\xi} \psi(\xi')\,\mathrm{e}^{-\xi'/c}\,\mathrm{d}\xi'\right]. \tag{149}$$

For $q(\xi)$ to be bounded as $\xi \to \infty$, we require the term in square brackets to vanish. This yields an implicit equation for the wave speed, c, in the form

$$\theta = \frac{1}{c}\int_0^{\infty}\left(\int_{\frac{\xi'}{1-c/v}}^{\infty} w(y)\,\mathrm{d}y\right)\mathrm{e}^{-\xi'/c}\,\mathrm{d}\xi'. \tag{150}$$

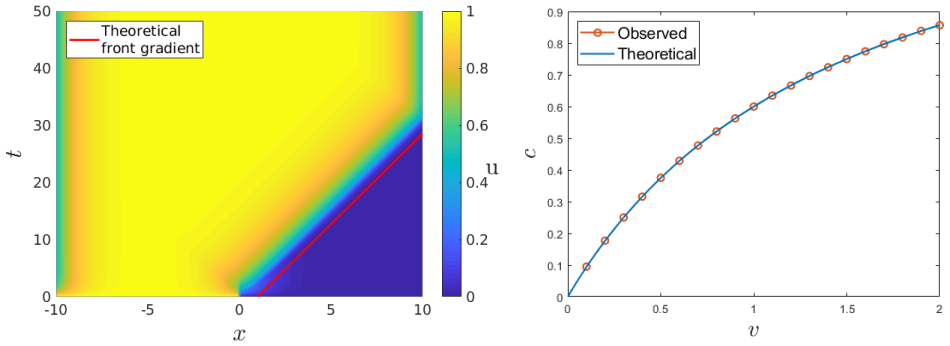
For the purpose of this example, we select the connectivity kernel to be the function $w(x) = \mathrm{e}^{-|x|}/2$. This allows Eq. (150) to be evaluated to give

$$\theta = \frac{1}{2}\frac{v - c}{c(v-1) + v}. \tag{151}$$

Rearranging for c, the speed of the travelling front solution is determined explicitly by the equation

$$c = \frac{v\,(2\theta - 1)}{2\theta - 1 - 2\theta v}. \tag{152}$$

Starting from an initial condition of $u\,(x, 0) = H\,(-x)$, we numerically solve Eq. (96) in the presence of axonal delays using the NFESOLVE library's *SparseDelayRungeKutta32Solver* solver, with the delay set up following the approach outlined in Section 5.5.1. From the numerical solution, the observed wave speed at which the front propagates may be calculated in order to compare it to the theoretical result given by Eq. (152).



(a) Example travelling front solution.

(b) Front speeds plotted against v.

Figure 26: Figure (a) depicts a travelling front solution to Eq. (96) on the domain $[-10, 10]$, with threshold $\theta = 0.2$ and axonal delay speed $v = 0.4$. Figure (b) shows the theoretical and observed travelling front speeds c plotted against the axonal delay speed v.

As the axonal delay speed, v, is increased, the speed at which the front propagates also increases. Asymptotically, for small values of v, the wave speed follows $c \sim v$. As v gets increasingly large, however, the wave speed conforms towards $c \sim (1 - 2\theta)/2\theta$. Figure 26a shows an example solution, along with the theoretical front propagation gradient, for a firing rate threshold value of $\theta = 0.2$ and an axonal delay speed of $v = 0.4$. As depicted in Figure 26b, a numerical analysis of the propagation speed of the front was undertaken to confirm whether the observed propagation speed matches

the theoretical wave speed given by Eq. (152). It is clear from Figure 26b that the theoretical and observed front speeds are extremely close together. The difference between the two lines is on the order of 10^{-3}. This gives confidence that the sparse delay solver suite of the NFESOLVE library yields numerical results that conform to the theory.

5.6 SUMMARY

The results presented in this chapter allow us to say with a high degree of confidence that the NFESOLVE library performs as expected and that there should be no problems when it is employed to solve more advanced models of neuronal activity. The convergence analyses undertaken clearly depict that the non-adaptive and adaptive solvers, for both ODEs and DDEs, all function as their named orders state. Users who wish to employ these schemes to solve differential equations in settings other than mathematical neuroscience should not face any issues when it comes to the library accurately solving their problems numerically. When it comes to the area of neural modelling, we have illustrated a number of examples, both theoretically and numerically, that are based on known results in mathematical neuroscience. From spherical geometries to NFEs with full-scale space-dependent delays, these results are important as it validates that the NFESOLVE library is capable of handling the models that typically form the basis of other more advanced models, some of which will be presented and explored later in this thesis, with a heavy emphasis on non-standard geometries and the incorporation of axonal delays.

6

NEURAL MASS NETWORKS

6.1 INTRODUCTION

Neural mass models are typically low dimensional models of neuronal activity that are employed to describe the dynamics of large populations of neurons. These are often extended to a network setting, where each node of the network represents a 'region' of the brain. The local dynamics are mediated by the neural mass model itself, while the node-to-node connections are described by means of a connectivity weight matrix. These weights stem naturally from the physical structure of the brain, i.e., regions that are strongly connected have more direct physical paths between them. The output of the models, often referred to as representing brain activity, can be post-processed to explore the functional relationship between signals. For example, if the signals emanating from two regions behave in a similar fashion, then they are said to be strongly functionally connected. With the extremely large amount of connectome data that is now available, one of the foci within the neural modelling community is incorporating real structural brain data into models of neuronal activity and comparing the output of the models with functional data.

In this chapter, we aim to explore the relationship between structural and functional connectivity, by means of neural mass modelling. This is based on work by Tewarie et al. [164] and Forrester et al. [65, 66]. To

start, we introduce the specific brain data that will be incorporated into the models. This includes structural and functional connectivity data, along with distance data for the white matter tracts that physically connect brain regions. Following this, we discuss how the analysis from Section 3.2 can be employed to engineer the output of a network model in order to best reflect the functional connectivity patterns observed in the data. Two models of neural activity are then presented, starting with the more traditional Wilson-Cowan model, and moving on to the next-generation neural mass model developed by Coombes and Byrne [36]. The benefits and limitations of these two models are explored, and finally, the results of each are compared to real functional connectivity data. This allows for an insight into how models of neuronal activity can be utilised to describe the observed relationships between structure and function.

6.2 USING STRUCTURE TO PREDICT FUNCTION

A key metric in understanding brain dynamics and patterns is functional connectivity (FC). As explained in Chapter 2 (Section 2.4), FC is based on the correlation of activity signals between different regions of the cortex. This is also discussed in more detail later in Section 6.2.1. There is a vast amount of data available, depicting not only the correlation of signals during resting state, but also during specific tasks that are designed to activate certain areas of the brain. One of the biggest challenges in the world of large-scale neural modelling is to accurately recreate empirical FC data through simulation. Although this can be attempted using an inverse modelling approach, such as regression modelling [161, 184], where the empirical FC data is analysed and utilised to inform an underlying model, this chapter seeks to explore a forward modelling approach via the incorporation of real brain data into existing models of neural activity to generate simulated FC data that can be compared to empirical FC data. We do this through the medium of neural mass modelling.

As neural mass models typically simulate brain activity across a number of brain regions, post-processing techniques can be applied to the output signals to determine a measure of correlation between the time series of each node, thereby generating a FC matrix. There are a number of different methods, such as Pearson correlation [134], amplitude envelope correction [23, 83], and mean phase coherence [126, 65], that can be used to determine the correspondences between time series. The simulated FC can then be compared to empirical FC data by means of a similarity measure, such as Pearson correlation or the Jaccard similarity [99]. This begs the question: can simulated FC generated via a mathematical model of neural function be engineered such that it gives as close an approximation as possible to empirical FC data? This chapter explores a possible method of doing this, in the form of structural eigenmode fitting, and studies the differences between the results yielded from two different models of neural activity; one of which is a purely phenomenological model, and the other which is more firmly based in observed biological processes.

6.2.1 *Data*

In this chapter, the models and simulations will incorporate the use of real brain data. All of the data used is provided by the Human Connectome Project (HCP) [172, 173] and is publicly available at `https://www.humanconnectome.org`. The methods behind the acquisition of this data are outlined in Section 2.4. Although the full data set contains a large amount of data from 1200 subjects, we only consider a small subset of the data in this work. We take the structural connectivity, functional connectivity, and path length data from 10 individual subjects in this data set and average them to yield a single matrix for each data type.

As neural mass networks are made up of "clusters" of neurons, they typically contain a relatively small number of network nodes. Neural field models, on the other hand, are posed on a continuum domain, but due to

the numerical routines employed to solve them they tend to be discretised into a large number of spatial nodes [10]. When working at the neural mass level, it is common to split a cortical domain up according to a parcellated brain atlas and use each region as a node in the network. There are many different atlases that are used amongst the brain imaging community, such as the Automated Anatomical Labeling (AAL) atlas [168], and the HCP Multi-Modal Parcellation (MMP) atlas [71]; however, in this chapter, we consider the 68 node parcellation given by the Desikan-Killiany (DK) atlas [50]. This atlas subdivides the cortex on MRI scans into gyral based regions of interest. We selected this parcellation in order to balance the computational cost of evolving the network equations with comparing the output to functional connectivity data.

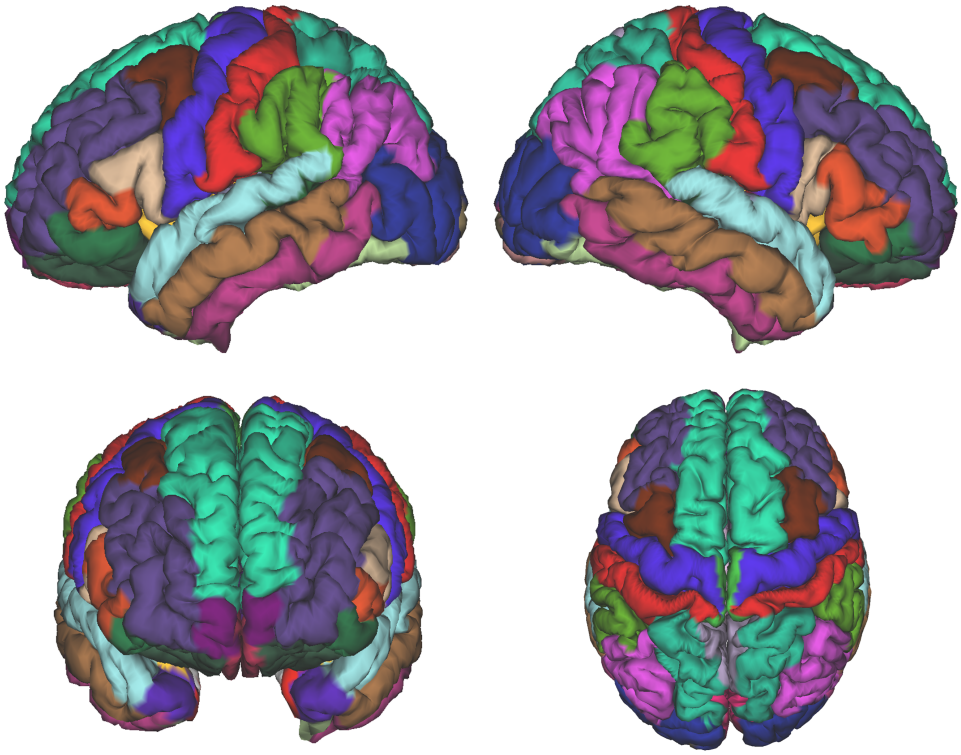


Figure 27: Visual representation of the Desikan-Killiany 68 region atlas on a full cortical domain, with each colour depicting a different region. The top row shows the view from the left and right sides of the cortex, while the bottom row shows the anterior and dorsal perspectives, respectively.

Structural Connectivity

As introduced in Section 2.4.1, a structural connectome is a comprehensive map of the physical connections between regions of the brain. An in-depth explanation of the how a connectome is constructed can be found in [159]. To recap, diffusion MRI is utilised to track the diffusion of water molecules through white matter tracts. As this diffusion is anisotropic, meaning that the water molecules travel along the axonal fibres rather than across them, post-processing techniques known as *tractography* can be applied to build a map of the white matter tracts in the brain. These algorithms work by taking an initial seed point and then either deterministically [12] or probabilistically [15] tracking the flow of the diffusion through areas of the white matter domain, known as voxels, until a boundary or other such termination point is reached. These flows are known as streamlines. Repeating this across multiple seed points yields multiple streamlines that estimate the bundles of axonal fibres. As the true structure is so complex, there is scope for these algorithms to produce false positive and false negative streamlines. Although dMRI cannot directly quantify connection strengths, it does allow for estimation of edge weights that can reflect desired properties, such as axonal density, myelination, etc. One of the most commonly used measures of connection strength is the streamline count, which is defined as the number of streamlines that intersect a pair of regions. Enumerating these counts for all pairs of regions allows for the population of a structural connectivity (SC) matrix.

For the structural connectivity matrix used in this chapter, connectome data was taken from 10 individual subjects and averaged to yield a single connectome. This was then divided through by the maximum element in the matrix in order to normalise the connection strength to be on $[0, 1]$. Manipulations via normalisation, or even non-linear transformations such as the logarithm, are common practice before analysis to reduce

the effects of different alogrithmic choices and ensure consistency across subjects [159, 73, 97].

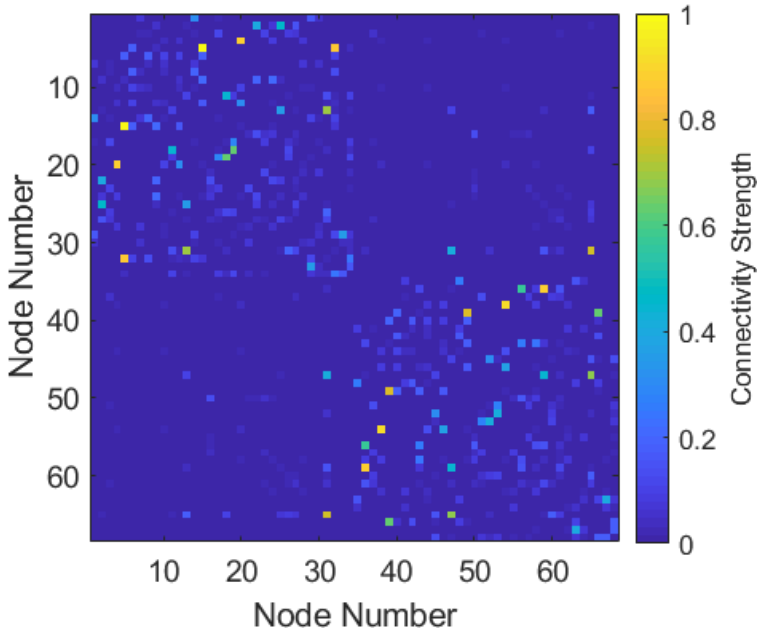


Figure 28: Structural connectivity matrix derived from connectome data averaged from 10 HCP subjects and normalised to be on $[0, 1]$.

As seen in Figure 28, the connectome is fairly sparse and does not contain a large degree of structure. This can be due to post-processing methods at the acquisition stage. To combat this, we consider applying a log transform to the data (before normalisation) of the form $w \to \log(w + A)$, where w is the connectivity matrix, A is some positive real constant, and log is the natural logarithm. This is also then renormalised to be on $[0, 1]$.

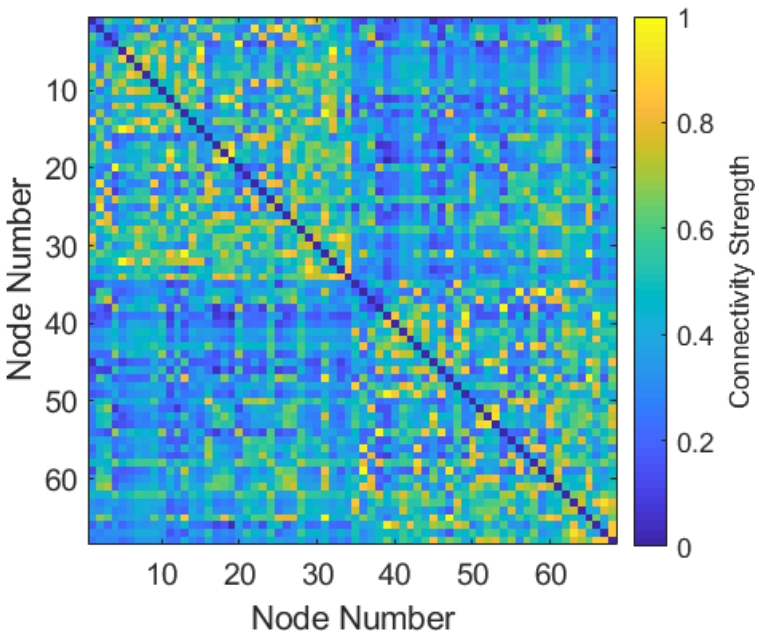


Figure 29: Log transformed connectivity data from Figure 28, with a transform of the form $\log(w + A)$, for $A = 0$. This is, again, followed by normalisation to the range $[0, 1]$.

Evidently, from Figure 29, applying the log transform with a value of $A = 0$ reveals much more structure in the network. As A gets larger, the normalised 'logged' matrix gets closer and closer to the original 'unlogged' data.

Functional Connectivity

An introduction to functional MRI is given in Section 2.4.2. To recap, this data is acquired by measuring the changes in blood flow in the brain. The oxygenation of the haemoglobin in the blood determines how sensitive it is to a magnetic field, with deoxygenated haemoglobin being far more sensitive than oxygenated haemoglobin. When neurons fire, the MRI machine detects the changes in blood flow via the oxygenation of the haemoglobin, which leads to a visualisation of the different areas of the brain that are concurrently active at any given time. This is known as the BOLD signal. Comparing the activity profile of these signals at different spatial points in the brain, via correlation measures such as Pearson's

correlation coefficient [134], allows conclusions to be made about how the different regions of the brain are functionally connected. This is typically represented as a matrix, analogously to SC.

The fMRI data used in this chapter is resting state activity readings from 10 individual HCP subjects, in the form of the BOLD signal, approximately spanning 52.8 minutes. In the same way as Demirtaş et al. [48], to process the data into a FC matrix the BOLD signal time series for each region of interest are first z-scored (quantifying how much the data varies from the mean in terms of a signed multiple of the standard deviation) and then the correlation between each pair of time series is computed via the Pearson correlation coefficient. Similarly to the structural connectivity data, the FC matrices from these 10 subjects are then averaged to give a single matrix.

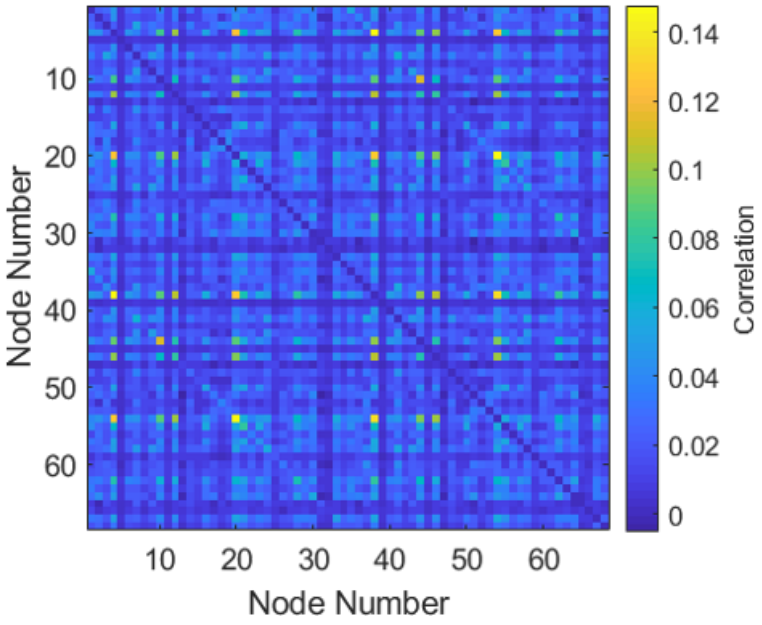


Figure 30: Functional connectivity matrix derived from data averaged from 10 HCP subjects.

Figure 30 shows a visual representation of this matrix, with areas of higher correlation coloured in yellow and areas of lower correlation

coloured in blue. This is the empirical FC data that we are aiming to replicate through simulation of neural mass models.

Distance Data

In order to incorporate axonal delays into the models, we assume that the delay time is the time taken for information to propagate along the axonal fibres between regions. To compute this, we require the lengths of the paths between each region. This data is acquired via the dMRI processes explained above in Section 6.2.1. Once again, the path lengths for 10 HCP subjects is averaged to give a single data set.

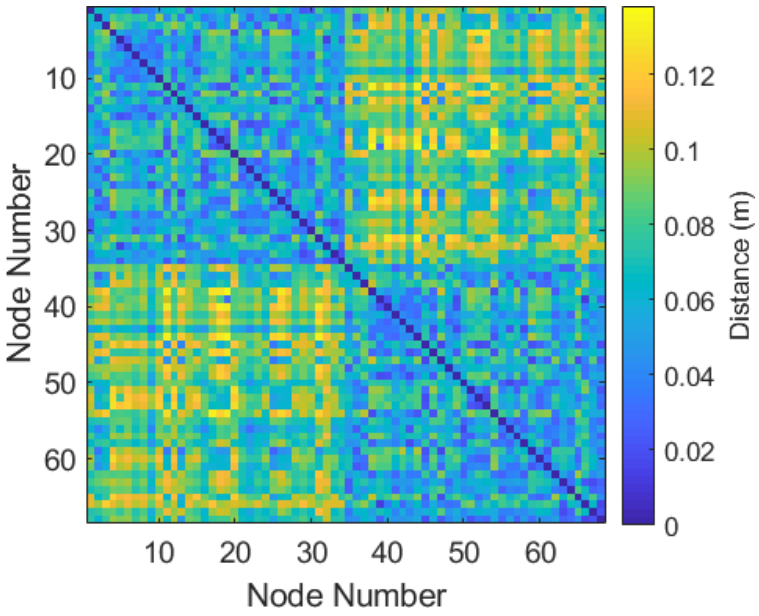


Figure 31: Path length data averaged from 10 HCP subjects, showing the average distance (in metres) between each region of the DK atlas.

As it is the intention to simulate with a time unit of seconds, the distance data is given in metres and we assume that the propagation velocity is in metres per second. Figure 31 visualises the data in the form of a matrix, similarly to the structural and functional connectivity data seen above.

6.2.2 *Eigenmode Fitting*

The structural connectivity matrices that are used in neural modelling play a huge part in governing the solutions and stability of the models that they are incorporated in, whether they are derived from real structural data, or are based entirely on a user-defined connectivity function. This can be used advantageously to engineer the simulation output so that it may conform to empirical FC data. As seen in Chapter 3 (Section 3.2), the steady state stability behaviour of an arbitrary non-linear network can be determined by the eigenvalues of the non-local connectivity matrix, w. When the real part of specific eigenvalues of the Jacobian of the linearised network equations become positive, thereby pushing the system over the instability threshold, the resulting simulation output is reflective of the corresponding structural eigenmodes. If a parameter set for a given model can be determined, such that it excites specific structural eigenmodes that explain empirical FC to a desired degree, then it is reasonable to assume that the model output will give a good approximation of empirical FC.

Tewarie et al. [164] use an approach where they consider whether an empirical FC matrix can be explained by a linear combination of the eigenmodes of the structural matrix. To build this prediction of the FC, matrices were generated from each eigenvector of the SC matrix by taking the outer product of the eigenvector with itself. A non-linear least squares fitting algorithm was then used to establish the linear combinations of these matrices that most accurately fit the empirical FC matrix. Another method for generating these matrices for each eigenvector is to consider the cosine of the phase differences, as seen in [44]. For a given eigenvector, v^p, of length N, the columns of the corresponding matrix, M^p, are given by

$$M_i^p = \cos\left(\pi\left(v^p - v_i^p\right)\right), \tag{153}$$

for $i = 1, \ldots, N$. The multiplication by π allows the values of the matrix to be in the range $[0, 1]$. This is the method that will be used in this thesis as simulation results appear to be better reflected by the cosine predictions than the outer product predictions.

In order to determine a set of eigenmodes that best predict the empirical FC, the single best-fitting mode is found first and then consecutively built upon by adding in one mode at a time, each time locating the next best-fitting mode. To measure the goodness of fit, the adjusted coefficient of determination (or adjusted R^2) at each stage is considered. This value is the proportion of the variance in the empirical FC that is explained by the structural eigenmode fit, and is adjusted to account for the change in number of modes. An R^2 of 0 shows that the fit explains none of the variance of the empirical FC, while an R^2 of 1 means the model is a perfect fit. For the data introduced in Section 6.2.1, the eigenmode fitting process reveals that although the 'logged' matrices depict a greater degree of structure, the R^2 of the fit decreases as the offset value, A, tends to 0 from above. Hence, we consider an offset value of $A = 1$ in order to balance the exposure of structure versus a higher R^2. As an example, the first 5 best-fitting eigenmodes of this matrix to the empirical FC data are the 11th, 39th, 3rd, 68th and 66th eigenmodes. This fit yields an R^2 value of 0.3575. We note here that the ordering of the eigenmodes is with respect to the sorted order of the eigenvalues of the SC matrix, with the 1st eigenvalue being the largest and the 68th being the smallest.

Although this approach is useful in highlighting the eigenmodes that best fit the empirical FC, it is completely independent of any models of neural activity. It must be noted that just because a model can generate solutions that give a good fit to the empirical FC, it does not necessarily mean that the model output is a good reflection of brain data, and vice versa.

6.3 A NETWORK OF WILSON-COWAN NODES

Before exploring a next-generation neural mass model, it is important to study the existing models of neuronal activity so as to have a base to compare to. One of the most prevalent models in the history of neural mass networks is the Wilson-Cowan model, as introduced in Chapter 2 (Section 2.3.1). Naturally, the Wilson-Cowan model supports oscillatory solutions, which can be interpreted as naive representations of brain rhythms. In this section, we introduce a two-population model of neuronal activity and explore the dynamics that arise when it is posed on a realistic brain atlas, with connectivity and distance data provided by the HCP.

6.3.1 *The Model*

In [1, 164], the authors consider a network of neural masses whose dynamics are governed by the two-population Wilson-Cowan model, given by

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix}\tau_E E_i(t)\\ \tau_I I_i(t)\end{bmatrix} = -\begin{bmatrix}E_i(t)\\ I_i(t)\end{bmatrix} + f\begin{pmatrix}w^{EE}E_i(t) + w^{EI}I_i(t) + P + \sum_{j=1}^{N} w_{ij}E_j(t)\\ w^{IE}E_i(t) + w^{II}I_i(t)\end{pmatrix}. \tag{154}$$

Here, E_i and I_i refer to the mean firing rates of the excitatory and inhibitory populations at brain location $i = 1, \ldots, N$, τ_a are time-scaling constants (where $a \in \{E, I\}$), P is a constant external excitatory input, f is the standard sigmoidal response function (see Section 2.3.2) used throughout this thesis (with steepness, μ, and threshold, θ), w^{ab} are the local coupling strengths between populations (where $a, b \in \{E, I\}$), and w_{ij} are the non-local white-matter connections from region i to region j. The inhibitory connections in this model are purely local, with only the excitatory connections affecting the non-local dynamics.

Incorporation of Delays

The model presented above does not contain delays; however, it can be extended to incorporate distance-dependent delays by changing the summation term in the first equation to

$$\sum_{j=1}^{N} w_{ij} E_j \left(t - \frac{D_{ij}}{v} \right), \tag{155}$$

where D_{ij} is some measure of distance between region i and region j, and v is the axonal conduction velocity. We make the assumption that v is constant and identical for all paths.

6.3.2 *Linear Stability Analysis*

In order for oscillations to arise in the solution, it is useful to locate a parameter regime where the system has undergone a Hopf bifurcation. Using the linear stability analysis of non-linear networks presented in Chapter 3 (Section 3.2), the eigenvalue problem for the network can be constructed in terms of the eigenvalues of the SC matrix, w.

Firstly, Eq. (154) can be written in the form

$$\frac{\mathrm{d}}{\mathrm{d}t} x_i = F(x_i) + G\left(w^{\mathrm{loc}} x_i + s_i + \rho \right), \quad s_i = \sum_{j=1}^{N} w_{ij} H(x_j), \tag{156}$$

with $F(x) = -\Gamma^{-1} x$, $G(x) = \Gamma^{-1} f(x)$,

$$x_i = \begin{bmatrix} E_i \\ I_i \end{bmatrix}, \; H(x) = \begin{bmatrix} E \\ 0 \end{bmatrix}, \; \Gamma = \begin{bmatrix} \tau_E & 0 \\ 0 & \tau_I \end{bmatrix}, \; w^{\mathrm{loc}} = \begin{bmatrix} w^{EE} & w^{EI} \\ w^{IE} & w^{II} \end{bmatrix}, \; \rho = \begin{bmatrix} P \\ 0 \end{bmatrix}.$$

Using this notation, the steady state of the network is given by the solution to

$$\bar{x}_i = f\left(w^{\mathrm{loc}} \bar{x}_i + \bar{s}_i + \rho \right), \quad \bar{s}_i = \sum_{j=1}^{N} w_{ij} H(\bar{x}_j). \tag{157}$$

From Eq. (55), we define the matrices

$$D\widetilde{F}_i = \Gamma^{-1}\left[-\mathcal{I}_2 + Df\left(w^{\text{loc}}\overline{x}_i + \overline{s}_i + \rho\right)w^{\text{loc}}\right], \tag{158}$$

$$D\widetilde{G}_i = \Gamma^{-1}Df\left(w^{\text{loc}}\overline{x}_i + \overline{s}_i + \rho\right)\begin{bmatrix}1 & 0\\ 0 & 0\end{bmatrix}, \tag{159}$$

where $\mathcal{I}_2$ is the 2×2 identity matrix and $Df(x)$ is the diagonal matrix whose diagonal entries are given by $[Df(x)]_{ii} = f'\left(x^i\right)$. Imposing a row-sum normalisation condition on the SC matrix ($\sum_j w_{ij} = 1$ for all $i = 1,\ldots,N$), the dependence on i is removed from $D\widetilde{F}_i$ and $D\widetilde{G}_i$, and the steady state $\overline{x}_i$ now becomes the homogeneous steady state $\overline{x}_i = \overline{x}$, where $\overline{x}$ is the solution to

$$\overline{x} = f\left(w^{\text{loc}}\overline{x} + H(\overline{x}) + \rho\right). \tag{160}$$

Letting the eigenvalues of w be denoted by β_p, $p = 1,\ldots,N$, the full spectrum of eigenvalues, λ, can be generated for the network by solving the eigenvalue problem $\mathcal{E}(\lambda;p) := \left|\lambda\mathcal{I}_2 - D\widetilde{F} - \beta_p D\widetilde{G}\right| = 0$. Writing $D\widetilde{F} + \beta_p D\widetilde{G}$ as a new matrix $\mathcal{A}(p)$, the eigenvalue equation can be written in the traditional form $|\lambda\mathcal{I}_2 - \mathcal{A}(p)| = 0$, where

$$\mathcal{A}(p) = \Gamma^{-1}\left[\mathcal{I}_2 - Df\left(w^{\text{loc}}\overline{x} + H(\overline{x}) + \rho\right)\left[w^{\text{loc}} + \begin{bmatrix}\beta_p & 0\\ 0 & 0\end{bmatrix}\right]\right], \tag{161}$$

for $p = 1,\ldots,N$. This has solutions $\lambda = \lambda_{\pm}(p)$, where

$$\lambda_{\pm}(p) = \frac{1}{2}\left[\text{Tr}\left(\mathcal{A}(p)\right) \pm \sqrt{\text{Tr}\left(\mathcal{A}(p)\right)^2 - 4\det\left(\mathcal{A}(p)\right)}\right]. \tag{162}$$

A Hopf bifurcation ($\lambda = i\omega$, for some $\omega \in \mathbb{R}\setminus\{0\}$) occurs when both $\text{Tr}\left(\mathcal{A}(p)\right) = 0$ and $\det\left(\mathcal{A}(p)\right) > 0$.

In the presence of delays, the eigenvalue problem becomes a transcendental equation under the replacement

$$\beta_p \to \beta_p(\lambda) = \sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij}\,\mathrm{e}^{-\lambda\frac{D_{ij}}{v}}\gamma_i^p\zeta_j^p, \tag{163}$$

where γ^p and ζ^p are the pth normalised left and right eigenvectors of w, respectively, such that they form a dual basis of the eigenspace of w. To solve the delay problem, the use of a numerical non-linear equation solver is required.

6.3.3 *Results*

The linear stability analysis presented in the previous section relies on the imposition of a row-sum normalisation condition on the SC matrix. This condition greatly aids in simplifying the search for a parameter regime that allows the system to undergo a Hopf bifurcation. Not only does it facilitate the existence of a homogeneous steady state, but it also transforms the size $2N \times 2N$ eigenvalue problem into N individual 2×2 eigenvalue problems (see Section 3.2 for a full explanation of why this is possible). This allows for the easy distinction between the eigenmodes of the SC matrix that are causing the system to become unstable. As discussed in Section 6.2.1, normalisations of this kind are commonly applied to structural connectome data to ensure consistency across the data of different subjects and reduce the effect of confounds reflecting algorithmic choices [159, 66].

To start, we consider the model in the absence of delays. For the SC matrix, we select the log transformed data (as explained in Section 6.2.1) with an offset value of $A = 1$. The core model parameter values, that remian unchanged for the rest of this section, are the same as those used by Tewarie et al. [164], who base their choices on previous work by Abeysuriya et al. [1] and Deco et al. [45]. These are detailed below in Table 2.

Parameter	Description	Value
τ_E	Excitatory time scale	0.01
τ_I	Inhibitory time scale	0.02
w^{EE}	Local excitatory-to-excitatory coupling	3.5
w^{EI}	Local inhibitory-to-excitatory coupling	-2.5
w^{IE}	Local excitatory-to-inhibitory coupling	3.75
w^{II}	Local inhibitory-to-inhibitory coupling	0.0
μ	Firing rate steepness	4.0
θ	Firing rate threshold	1.0

Table 2: Wilson-Cowan network parameters, based on those used in [164].

This leaves the constant excitatory drive parameter, P, which can be tuned to cause the system to undergo a Hopf bifurcation, leading to oscillatory solutions. In the absence of delays, the order of the eigenvalues of the SC matrix ($\beta_1 > \beta_2 > \ldots > \beta_N$) is preserved by the eigenvalues of the system, meaning that for each complex conjugate pair of eigenvalues, $\lambda_\pm$, the real part follows $\lambda_\pm(1) > \lambda_\pm(2) > \ldots > \lambda_\pm(N)$. No matter what parameters are selected, this order is always preserved, thereby resulting in the first eigenmode of the SC matrix always crossing the imaginary axis first. This is illustrated in Figure 32, which depicts the entire spectrum of eigenvalues for a variety of values of P.

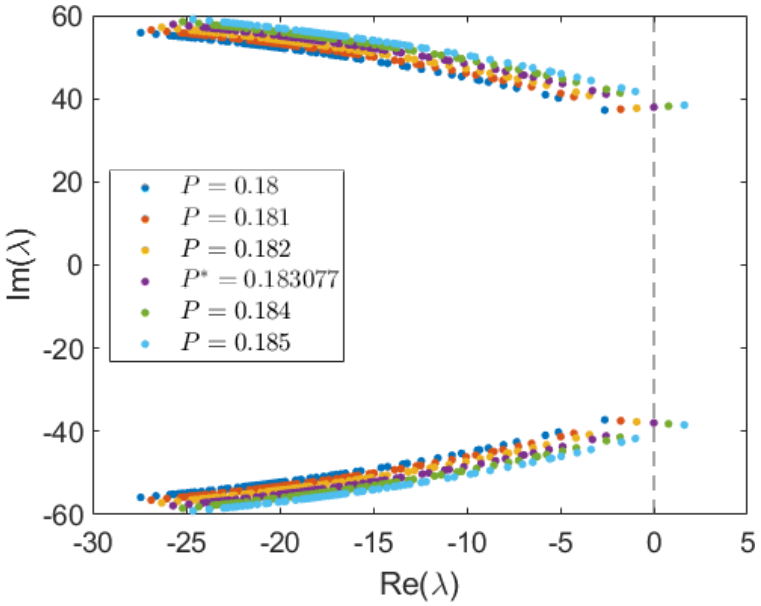


Figure 32: Eigenvalue spectrum of the Wilson-Cowan network in the absence of delays, computed around the steady state, $\overline{x}$, for a range of values of the excitatory drive parameter, P. The conjugate pair of eigenvalues corresponding to the first eigenmode of the SC matrix cross the imaginary axis at the critical value of $P^* = 0.183077$, resulting in a Hopf bifurcation.

It is evident that as P is increased, the real part of the spectrum also increases. At the critical value of $P^* = 0.183077$, the spectrum crosses the imaginary axis, led by the conjugate pair of eigenvalues corresponding to the first eigenmode of the SC matrix. This means that it is impossible to excite any specific eigenmodes that might yield a better simulation output, as predicted by the methods outlined in Section 6.2.2.

In order to allow a greater degree of flexibility when it comes to exciting different eigenmodes, axonal delays can be introduced into the system. Eq. (155) shows the typical format of these distance-dependent delays; however, it has been shown by Tewarie et al. [164] that very little change to the ordering of the eigenmodes occurs unless an additional

constant offset delay is also introduced. This results in the delay term now taking the form

$$\sum_{j=1}^{N} w_{ij} E_j \left(t - \left(\tau_0 + \frac{D_{ij}}{v} \right) \right), \tag{164}$$

where τ_0 is the offset delay. Ideally, this offset delay would not be present in the model as it is not as physically relevant (as explained in Section 2.3.2). Some authors attribute it to representing a synaptic and dendritic processing delay [125], though it is typically incorporated in firing rate models as it can allow for dynamics similar to those seen in simulations of large-scale spiking networks [142]. We select the axonal conduction velocity, v, to be $10\,\mathrm{m\,s^{-1}}$, in line with [164, 45].

For this system of equations, as the offset delay is increased, it acts to rotate the spectrum respective to each half of the imaginary plane. An example of this can be seen in Figure 33.

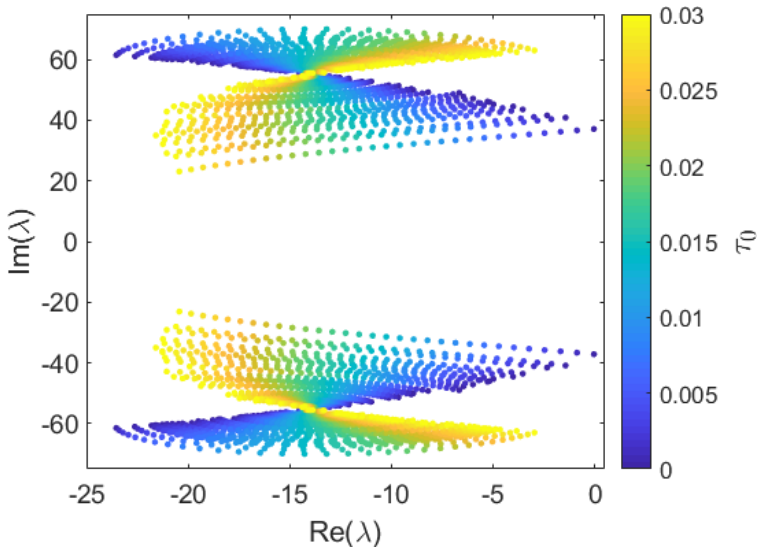


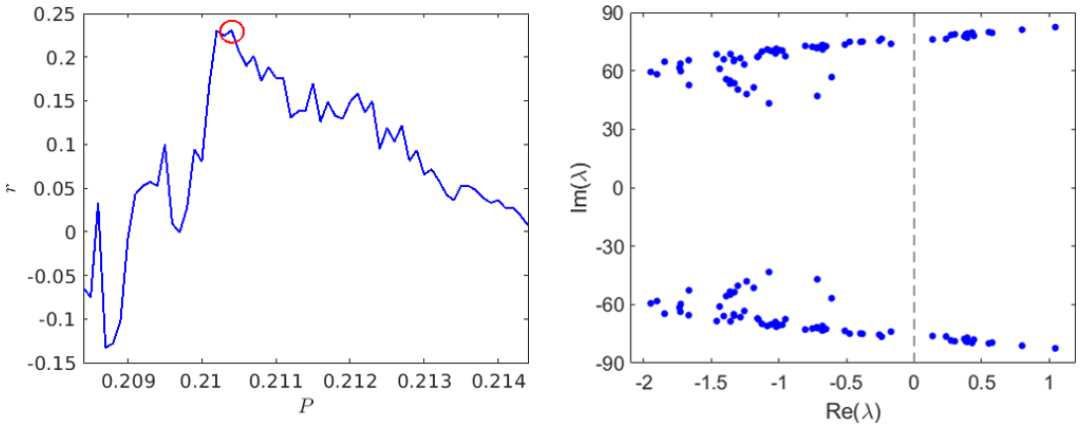
Figure 33: Eigenvalue spectrum of the Wilson-Cowan network with delays, computed around the steady state, $\bar{x}$, for a range of values (between 0 and 0.03 seconds) of the constant offset delay, τ_0. The excitatory drive parameter, P, is fixed with value $P = 0.1868$.

While this still restricts the ability to excite any specifically chosen eigenmodes, it does allow the system to be excited from the opposite end of the

spectrum to the non-delay case, yielding more opportunity for finding parameter regimes that generate solutions which emulate empirical FC data. As stated in Section 6.2.2, the 68th and 66th eigenmodes are both in the list of the top five best-fitting eigenmodes, therefore τ_0 can be utilised to push these eigenmodes unstable, amongst others at the lower end of the spectrum. It should be noted that the linear theory only provides a prediction of the behaviour close to the point of bifurcation. Due to the nature of the spectrum, it is extremely difficult, if not impossible, to simultaneously excite just the eigenmodes indicated by the fitting prediction. As the system is pushed further away from bifurcation, the linear theory breaks down and yields little information as to how the unstable eigenmodes will interact with each other.

To find a parameter regime that yields the results that best reflect the empirical FC, we first fix the offset delay, τ_0, at a point where the spectrum is rotated enough to excite the system from the lower end of the spectrum. An appropriate value for which this occurs is $\tau_0 = 0.013$s, also leading to the magnitude of the imaginary part of the leading eigenvalues to be around 80. This yields oscillations that have a frequency of around 12Hz, which sits on the upper end of the alpha band. Next, slowly increasing the excitatory drive parameter, P, over a range of values, we generate a simulation output for each value. This is done efficiently by utilising the *DelaySparseRungeKutta32Solver* solver from the NFESOLVE suite. As P is increased, more and more eigenvalues become unstable. The amplitude envelope correlation (AEC) [23] can then be computed, for a variable of choice, to generate FC matrices for each data set. This is done by first computing the envelope of each node's signal by taking the absolute value of the analytic signal, which we compute by making use of MATLAB's built-in *hilbert* function. Following this, a Pearson correlation is performed between each pair of envelopes to build a FC matrix. For each P value, we compare the outputted AEC FC matrix to the empirical FC data by means of a Pearson

correlation. Figure 34a shows this Pearson correlation coefficient, r, for each value of P, where the variable of interest is the excitatory firing rate, E.



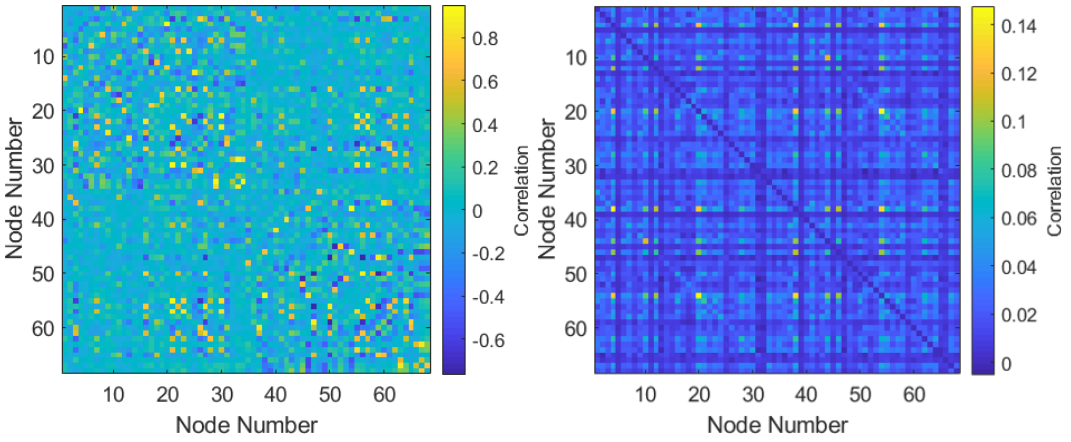
(a) Pearson correlation of the AEC of the model output compared to empirical FC.

(b) Spectrum of eigenvalues for the delayed Wilson-Cowan network with $P = 0.2104$, computed around the steady state, $\bar{x}$.

Figure 34: On the left, Figure (a) depicts the Pearson correlation coefficient, r, for the comparison of the AEC of the simulated excitatory firing rate (E) output signals against the empirical FC, computed for a range of values of the excitatory drive parameter, P, and for the fixed offset delay value $\tau_0 = 0.013$s. Highlighted in the red circle is the maxima of the plot, indicating that a value of $P = 0.2104$ yields the simulation results that most emulate the empirical FC. Figure (b) shows the eigenvalue spectrum for this value of P, computed around the steady state, $\bar{x}$.

The value of P for which the simulation output yields the closest match to empirical FC occurs at $P = 0.2104$, as indicated by the red circle, with the Pearson r value being $r = 0.2307$. The entire spectrum for this parameter regime is depicted in Figure 34b. The unstable eigenmodes of the system are led by 68th eigenmode and tailed by the 53rd eigenmode, with the exclusion of the 54th mode which has not yet crossed the imaginary axis. The eigenmodes in between are not necessarily ordered consecutively. Although

the linear theory does not show how these modes interact, it is clear that in order to get a simulation output that is most reflective of the empirical data, a great deal of modes must be pushed unstable and far from bifurcation.



(a) Simulated FC matrix.

(b) Empirical FC matrix.

Figure 35: Figure (a) shows the simulated FC matrix for the excitatory firing rate variable E, generated via AEC, with parameters $P = 0.2104$, $\tau_0 = 0.013$s and other parameters given in Table 2. On the right, Figure (b) shows the empirical FC matrix, as introduced in Section 6.2.1. The Pearson r value that measures the correlation between the two matrices, as shown in Figure 34, is $r = 0.2307$.

The simulated FC matrix can be seen in Figure 35, alongside the empirical FC matrix from Section 6.2.1. Although there are aspects of the empirical FC that come through in the simulated data, it is evident that the Wilson-Cowan model clearly has its limitations when it comes to producing output that is reflective of observed functional connectivity patterns.

6.4 NEXT-GENERATION NEURAL MASS MODEL

Models such as the Wilson-Cowan model have been around since the 1970s and have been employed in a number of different settings to simulate brain activity. However, the explanatory power of such models can only go so far.

Recent developments in neural modelling have led to more advanced and descriptive models emerging in the community. The next-generation model that we introduce in this section is based on work pioneered by Coombes, Byrne, and others [36, 25, 26, 28, 29]. They use an approach that has been explored by Montbrió et al. [124] to reduce a network of quadratic integrate-and-fire (QIF) neurons in the thermodynamic limit, using the so-called Ott-Antonsen ansatz [131]. This work is taken further by Byrne and Coombes by adapting the model to include intrinsic biological mechanisms such as synaptic reversal potentials and gap-junction coupling at the local nodal level, and axonal delays at the larger spatial scale. This model has already been used to describe phenomena such as beta rebound [27], whereby there is an observed decrease in beta band oscillations during body movement, followed by an increase above baseline once movement has ceased, before settling back to the default state.

It is our aim in this section to explore the model in a network setting and to replicate empirical functional connectivity results, investigating the differences when compared to the Wilson-Cowan model seen in the previous section (Section 6.3).

6.4.1 *The Model*

Chemical synapses, as described in Chapter 2 (Section 2.1), allow for the transmission of currents between neurons. These currents are best modelled using event-driven interactions. Denoting the mth firing time of the jth neuron by T_j^m, the current received by a given neuron, i, from neuron j is proportional to $\sum_{m\in\mathbb{Z}} s\left(t - T_j^m\right)$. Here s is the temporal shape of the post-synaptic response generated by the firing event. Typically, this is chosen to be a Green's function of a linear differential operator Q, thereby satisfying the equation $Qs = \delta$, where δ is the Dirac delta function. Although there

are several common choices of s (as explored by Byrne et al. in [25, 27]), here we consider the α-function response given by

$$s(t) = \alpha^2 t \exp(-\alpha t) H(t), \tag{165}$$

where $H(t)$ is the Heaviside step function. An example of this curve with various α values is depicted in Figure 36.

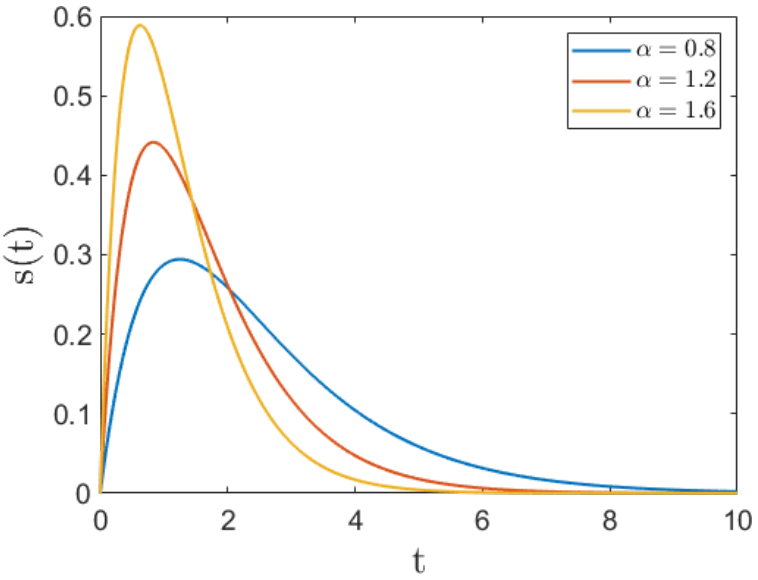


Figure 36: Example of the α-function response curve given by Eq. (165), with three different values of α.

This choice in s yields the corresponding linear differential operator

$$Q = \left(1 + \frac{1}{\alpha}\frac{\mathrm{d}}{\mathrm{d}t}\right)^2, \tag{166}$$

where α^{-1} is the time-to-peak of the synaptic response.

On the other hand, electrical synapses allow for the post-synaptic neuron to directly "feel" the shape of the action potential being sent by the pre-synaptic neuron. As outlined in Chapter 2 (Section 2.1), this is facilitated by conductive channels called gap junctions. Following a simple ohmic model, for two neurons with voltages v_i and v_j, the current flowing between neuron i and j is proportional to the difference in voltage between the post- and pre-synaptic neuron, i.e., $v_j - v_i$.

At the microscopic level, consider a network of N QIF neurons with gap junction coupling, synaptic coupling, and synaptic reversal potentials. The voltage, v_i, of the ith neuron, $i = 1, \ldots, N$, with inputs from the rest of the network, is described by the system

$$\tau \frac{\mathrm{d}v_i}{\mathrm{d}t} = \eta_i + v_i^2 + \frac{\kappa^v}{N} \sum_{j=1}^{N} \left(v_j - v_i\right) + I_i, \tag{167}$$

where τ is the membrane time constant, η_i are constant background drives, κ^v is the gap junction coupling strength, and I_i is a synaptic input current. This current is assumed to take the form $I_i = g\,(t)\left(v_{\text{syn}} - v_i\right)$, where v_{syn} is the synaptic reversal potential of the neurons, and g is the overall conductance. The synaptic reversal potential is the value of the membrane potential for which there is no net transfer of ionic current from one side of the post-synaptic membrane to the other. As the cell receives and transmits signals, it seeks to bring its voltage to an equilibrium of this value. If the synaptic current I_i is positive (negative) then the synapse is referred to as excitatory (inhibitory). The global conductance, g, is mediated by the equation

$$Qg = \frac{\kappa^s}{N} \sum_{j=1}^{N} \sum_{m \in \mathbb{Z}} \delta\left(t - T_j^m\right). \tag{168}$$

Here, Q is the linear differential operator given by Eq. (166), and κ^s is the synaptic coupling strength. This choice of Q is to best capture the temporal characteristics of the synaptic response. The mth firing time of the jth neuron is defined implicitly by $v_j\left(T_j^m\right) = v_{\text{th}}$. The network neurons are subject to reset $v_i \to v_{\text{reset}}$ at the firing times T_i^m. The threshold and reset values are considered to be $v_{\text{th}} \to \infty$ and $v_{\text{reset}} \to -\infty$. The drives, η_i, are selected such that they are randomly drawn from a Lorentzian distribution

$$L(\eta) = \frac{1}{\pi} \frac{\Delta}{\left(\eta - \eta_0\right)^2 + \Delta^2}, \tag{169}$$

with center η_0 and half-width Δ. These two parameters can be thought of as setting the level of excitability and the degree of heterogeneity in the

network, respectively.

Using the methods outlined in [124, 29], we consider the network in the thermodynamic limit. As $N \to \infty$, the state of the network at time t can be described by a continuous probability density function $\rho(v|\eta, t)$. This satisfies the continuity equation

$$\frac{\partial \rho}{\partial t} + \frac{\partial (\rho \dot{v})}{\partial v} = 0, \tag{170}$$

which arises from the conservation of oscillators. A realisation of the system is

$$\tau \dot{v} = \eta + v^2 - \kappa^v v + \kappa^v V + g\left(v_{\text{syn}} - v\right), \tag{171}$$

$$Qg = \kappa^s R, \tag{172}$$

where V is the average voltage, given by

$$V(t) = \lim_{N\to\infty} \frac{1}{N} \sum_{j=1}^{N} v_j, \tag{173}$$

and R is the population firing rate, given by

$$R(t) = \lim_{N\to\infty} \frac{1}{N} \sum_{j=1}^{N} \sum_{m\in\mathbb{Z}} \delta\left(t - T_j^m\right). \tag{174}$$

Now, using a Lorentzian Ansatz [124], we assume that $\rho(v|\eta, t)$ has a solution of the form

$$\rho(v|\eta, t) = \frac{1}{\pi} \frac{x(\eta, t)}{(v - y(\eta, t))^2 + x(\eta, t)^2}. \tag{175}$$

For a fixed η, the firing rate (number of spikes per unit time) can be calculated as

$$r(\eta, t) = \rho(v \to \infty|\eta, t)\, \dot{v}(v \to \infty|\eta, t). \tag{176}$$

This yields the result

$$x(\eta, t) = \pi\tau r(\eta, t). \tag{177}$$

After averaging over the distribution of single neuron drives, $L(\eta)$, the total population firing rate can be written as

$$R(t) = \frac{1}{\pi\tau} \int_{-\infty}^{\infty} x(\eta, t)\, L(\eta)\, \mathrm{d}\eta. \tag{178}$$

Similarly, we gain an expression for $V(t)$ by exploiting the pole structure of Eq. (175) and taking the contour integral of $v\rho(v|\eta, t)$ over v, then once again averaging over $L(\eta)$. This yields

$$V(t) = \int_{-\infty}^{\infty} y(\eta, t) L(\eta) \, \mathrm{d}\eta. \tag{179}$$

For fixed η, substituting Eq. (175) into the continuity equation, Eq. (170), and balancing powers of v shows that x and y obey two coupled differential equations that can be written as

$$\tau \dot{w} = -(g + \kappa^v) w + i\left(\eta + \kappa^v V + v_{\text{syn}} g - w^2\right), \tag{180}$$

where $w(\eta, t) = x(\eta, t) + iy(\eta, t)$. Finally, using the fact that $L(\eta)$ has poles at $\eta_\pm = \eta_0 \pm i\Delta$, the expressions for $R(t)$ and $V(t)$ can be evaluated using contour integration to give

$$R(t) = \frac{1}{\pi\tau} x(\eta_-, t), \tag{181}$$

$$V(t) = y(\eta_-, t). \tag{182}$$

This allows for the substitution of $w(\eta_-, t) = \pi\tau R(t) + iV(t)$ into Eq. (180), yielding

$$\begin{aligned} \pi\tau^2 \dot{R} + i\tau \dot{V} = &-(g + \kappa^v) \pi\tau R + \Delta + 2\pi\tau RV \\ &+ i\left(-(g + \kappa^v) V + \eta_0 + \kappa^v V + v_{\text{syn}} g - \pi^2\tau^2 R^2 + V^2\right). \end{aligned} \tag{183}$$

Simplifying and balancing real and imaginary terms, the equations for the next-generation neural mass model with gap junction coupling, synaptic coupling, and synaptic reversal potentials, are given by

$$\tau \dot{R} = 2RV + \frac{\Delta}{\pi\tau} - R(g + \kappa^v), \tag{184}$$

$$\tau \dot{V} = \eta_0 - \pi^2\tau^2 R^2 + V^2 + g(v_{\text{syn}} - V), \tag{185}$$

$$Qg = \kappa^s R. \tag{186}$$

Multiple Interacting Sub-Populations

Many models, such as the Wilson-Cowan equations (Eq. (154)), consider multiple sub-populations of neurons. Most commonly, an excitatory population and an inhibitory population. This can be incorporated within the

next-generation model by generalising the mean-field equations (Eqs. (184) to (186)) to treat electrical connections between sub-populations E (excitatory) and I (inhibitory), giving the equations

$$\tau_a \dot{R}_a = 2R_a V_a + \frac{\Delta_a}{\pi\tau_a} - R_a \sum_{b\in\{E,I\}} \left(g_{ab} + \kappa^v_{ab}\right), \tag{187}$$

$$\begin{aligned}\tau_a \dot{V}_a = \eta^a_0 - \pi^2\tau_a^2 R_a^2 + V_a^2 \\ + \sum_{b\in\{E,I\}} \kappa^v_{ab}\left(V_b - V_a\right) + \sum_{b\in\{E,I\}} g_{ab}\left(v^{ab}_{\text{syn}} - V_a\right),\end{aligned} \tag{188}$$

$$Q_{ab} g_{ab} = \kappa^s_{ab} R_b, \tag{189}$$

where $a, b \in \{E, I\}$, $Q_{ab} = \left(1 + \alpha^{-1}_{ab}\,\mathrm{d}/\mathrm{d}t\right)^2$, and the subscript ab dictates that b is the pre-synaptic sub-population and a is the post-synaptic sub-population. Each sub-population, a, has its own time scale, τ_a, and background drives drawn from a Lorentzian distribution with center η^a_0 and half-width Δ_a. There are now three gap junction coupling strengths (as $\kappa^v_{EI} = \kappa^v_{IE}$), four synaptic coupling strengths, four synaptic reversal potentials, and four synaptic time scales, that mediate the interactions between populations.

Networks of Populations

The mean-field model (Eqs. (184) to (186)) characterises the dynamics of a single population of neurons. This can be extended to describe an interconnected network of N populations, which can be attributed to different areas of the brain. Let i denote the ith population (or 'node') in the network, where $i = 1, \ldots, N$. The dynamics of each node is governed by the original equations; however, to facilitate the long-range connections between

nodes, an additional g variable, named g_{ext}, is introduced. The model now becomes

$$\tau \dot{R}_i = 2R_i V_i + \frac{\Delta}{\pi\tau} - R_i \left(g_i + g_{\text{ext}} + \kappa^v \right), \tag{190}$$

$$\tau \dot{V}_i = \eta_0 - \pi^2 \tau^2 R_i^2 + V_i^2 + \left(g_i + g_{\text{ext}} \right) \left(v_{\text{syn}} - V_i \right), \tag{191}$$

$$Q g_i = \kappa^s R_i, \qquad Q_{\text{ext}} g_{\text{ext}} = \kappa^s_{\text{ext}} \sum_{j=1}^{N} w_{ij} R_j, \tag{192}$$

where $Q_{\text{ext}} = \left(1 + \alpha_{\text{ext}}^{-1} \, \mathrm{d}/\mathrm{d}t\right)^2$, κ^s_{ext} is the non-local synaptic coupling strength, and w_{ij} are the entries of the non-local connectivity matrix w. Typically, when working with real connectivity data, the self-to-self connections in the matrix are zeroed. Hence, the self-to-self connections are governed by the evolution equations for g_i, while the external variable g_{ext} is introduced to govern the non-local dynamics.

Analogously, the excitatory and inhibitory sub-population mean field model, given by Eqs. (187) to (189), can also be extended to a network of N interacting sub-populations by introducing the non-local interaction variable, g_{ext}, along with its corresponding parameters α_{ext} and κ^s_{ext}. In this case, the long-range connections are considered to only apply to the excitatory population. The evolution equation of g_{ext} is dependent on the excitatory population firing rate, R_E.

As we wish to compare results to the Wilson-Cowan network introduced in Section 6.3, the excitatory-inhibitory network is the version of the next-generation model that we will be exploring in this section.

Incorporation of Delays

Similarly to the Wilson-Cowan network in Section 6.3.1, delays can be incorporated into the network by altering the long-range connection term. For the next-generation model, the long-range connections are mediated by the

evolution equation of the external variable g_{ext}, as shown in Eq. (192). With the inclusion of delays, this equation becomes

$$Q_{\text{ext}} g_{\text{ext}} = \kappa^s_{\text{ext}} \sum_{j=1}^{N} w_{ij} R_j \left(t - \frac{D_{ij}}{v} \right), \tag{193}$$

where, as in Section 6.3.1, D_{ij} is some measure of distance between node i and node j, and v is the axonal conduction velocity. Note that here we do not include the biologically unrealistic offset delay (τ_0) that was present in the Wilson-Cowan model.

6.4.2 *Linear Stability Analysis*

As explained previously, to best replicate brain data we require oscillatory solutions to arise from the model. A simple way to do this is to consider the homogeneous steady state of a system and use the linear stability techniques seen throughout this thesis to locate a Hopf bifurcation. Here, we apply these techniques and present an analysis of the next-generation model, starting with the single population model. This analysis is then extended to the network in a similar fashion to the Wilson-Cowan model (Section 6.3.2), allowing for the overall system to be reduced into N independent systems, each of which are representative of an eigenmode of the long-range connectivity matrix, w.

Single Population Model

For the single population mean-field model, described by Equations (184) to (186), the steady state of the system, given by the triple $\left(\overline{R}, \overline{V}, \overline{g}\right)$, is the solution to the non-linear equations

$$0 = 2\overline{R}\,\overline{V} + \frac{\Delta}{\pi\tau} - \overline{R}\left(\kappa^s \overline{R} + \kappa^v\right), \tag{194}$$

$$0 = \eta_0 - \pi^2\tau^2\overline{R}^2 + \overline{V}^2 + \kappa^s \overline{R}\left(v_{\text{syn}} - \overline{V}\right), \tag{195}$$

along with $\overline{g} = \kappa^s \overline{R}$. As R is a firing rate, it should be noted that it is required to be positive.

Consider the solution, $(R, V, g) = (\overline{R}, \overline{V}, \overline{g}) + (R_0, V_0, g_0)\, e^{\lambda t}$, arising from a small perturbation to the steady state. Firstly, substituting into Eq. (186) and linearising around the steady state yields

$$\left(1 + \frac{\lambda}{\alpha}\right)^2 g_0 = \kappa^s R_0, \tag{196}$$

making note that this can be rearranged to give $g_0 = \kappa^s R_0 \left(1 + {}^{\lambda}/_{\alpha}\right)^{-2}$. Substituting into the remaining equations, Eqs. (184) and (185), and again linearising around the steady state, gives the two dimensional system of equations

$$A(\lambda) \begin{pmatrix} R_0 \\ V_0 \end{pmatrix} = \begin{pmatrix} -\overline{R} g_0 \\ \left(v_{\text{syn}} - \overline{V}\right) g_0 \end{pmatrix}, \tag{197}$$

where $A(\lambda) = \tau\lambda \mathcal{I}_2 - J$, and J is the Jacobian matrix of Eqs. (184) and (185) evaluated at the steady state. Using the fact that $\overline{g} = \kappa^s \overline{R}$, this matrix is given by

$$J = \begin{pmatrix} 2\overline{V} - \kappa^s \overline{R} - \kappa^v & 2\overline{R} \\ -2\pi^2 \tau^2 \overline{R} & 2\overline{V} - \kappa^s \overline{R} \end{pmatrix}. \tag{198}$$

Applying Cramer's rule to Eq. (197) yields the solution

$$R_0 = \frac{1}{|A(\lambda)|} \begin{vmatrix} -\overline{R} g_0 & -2\overline{R} \\ \left(v_{\text{syn}} - \overline{V}\right) g_0 & \tau\lambda - 2\overline{V} + \kappa^s \overline{R} \end{vmatrix} \tag{199}$$

$$= \frac{g_0}{|A(\lambda)|} \begin{vmatrix} -\overline{R} & -2\overline{R} \\ \left(v_{\text{syn}} - \overline{V}\right) & \tau\lambda - 2\overline{V} + \kappa^s \overline{R} \end{vmatrix} \tag{200}$$

$$= \frac{g_0}{|A(\lambda)|} \overline{R} \left(\tau\lambda + \kappa^s \overline{R} - 2v_{\text{syn}}\right). \tag{201}$$

Using the earlier result that $g_0 = \kappa^s R_0 \left(1 + {}^{\lambda}/_{\alpha}\right)^{-2}$, and requiring that R_0 has a non-trivial solution, we may simplify further to give the eigenvalue equation

$$\mathcal{E}(\lambda) := |A(\lambda)| \left(1 + \frac{\lambda}{\alpha}\right)^2 + \kappa^s \overline{R} \left(\tau\lambda + \kappa^s \overline{R} - 2v_{\text{syn}}\right) = 0. \tag{202}$$

Expanding this out fully yields a fourth order polynomial equation in λ. Assuming that $\lambda = \nu + i\omega$, where $\nu, \omega \in \mathbb{R}$, a Hopf bifurcation occurs

when $v = 0$ and $\omega \neq 0$. To locate a Hopf bifurcation, we must solve the simultaneous equations

$$\mathrm{Re}\left(\mathcal{E}\left(i\omega\right)\right) = 0, \tag{203}$$

$$\mathrm{Im}\left(\mathcal{E}\left(i\omega\right)\right) = 0, \tag{204}$$

for $\omega \neq 0$. If $\omega = 0$, then the system undergoes a fold bifurcation; however, this does not lead to dynamic solutions.

E-I Sub-Population Network Model

The analyses seen thus far (in Sections 3.2 and 6.3.2) can be extended to the two sub-population network in order to generate eigenvalue spectra for the model. As the evolution equations for the variables g_{ab}, where $a, b \in \{E, I\}$, are second order, we introduce dummy variables h_{ab} for each equation, such that

$$\left(1 + \frac{1}{\alpha_{ab}}\frac{\mathrm{d}}{\mathrm{d}t}\right) g_{ab} = h_{ab}, \tag{205}$$

$$\left(1 + \frac{1}{\alpha_{ab}}\frac{\mathrm{d}}{\mathrm{d}t}\right) h_{ab} = \kappa_{ab}^{s} R_b, \tag{206}$$

which reduces the second order ODEs, $Qg_{ab} = \kappa_{ab}^{s} R_b$, into two first order ODEs for each ab pair. This is also applied to the external variable, g_{ext}, that mediates the non-local connections. After reducing the second order equations, the system is now made up of 14 first order ODEs. This system can be written generally in the form

$$\frac{\mathrm{d}}{\mathrm{d}t} x_i = F\left(x_i\right) + \sum_{j=1}^{N} w_{ij} G\left(x_j\right), \quad i = 1, \ldots, N, \tag{207}$$

where $x_i \in \mathbb{R}^{14}$, which is similar to that seen in Section 3.2. However, as the coupling between sub-populations in the next-generation model is mediated slightly differently, we use this altered form that does not include the w^{loc} matrix. Here, $x = (R_E, V_E, R_I, V_I, g_{EE}, g_{EI}, g_{IE}, g_{II}, g_{\mathrm{ext}}, h_{EE}, h_{EI}, h_{IE}, h_{II}, h_{\mathrm{ext}})$. The first four terms of $F\left(x\right)$ are given by the right-hand side of Eqs. (187) and (188),

under the replacement $g_{EE} \rightarrow g_{EE} + g_{\text{ext}}$. The remaining terms come from Eqs. (205) and (206), along with the same equations for g_{ext} and h_{ext}, except for the non-local connections term in the last equation, which is handled in G. This is given by $G(x) = (0,0,0,0,0,0,0,0,0,0,0,0,0,0,\kappa^{s}_{\text{ext}} R_E)$. Applying the linear stability analysis techniques explored in Section 3.2, the eigenvalue equation can be reduced down to N independent 14×14 determinant problems of the form

$$\mathcal{E}(\lambda; p) := \left|\lambda \mathcal{I}_{14} - DF - \beta_p DG\right| = 0, \quad p = 1, \ldots, N, \tag{208}$$

where DF and DG are the corresponding Jacobian matrices of F and G, and β_p, $p = 1, \ldots, N$, are the eigenvalues of the non-local connectivity matrix, w. Again, a Hopf bifurcation is located by solving Eqs. (203) and (204) for Eq. (208), for any $p = 1, \ldots, N$.

Analogously to the Wilson-Cowan model, the introduction of delays into the next-generation model, shown by Eq. (193), can be accounted for in the stability equation by the replacement $\beta_p \rightarrow \beta_p(\lambda)$, depicted in Eq. (163).

6.4.3 *Results*

In order to compare the results of the next-generation model to the Wilson-Cowan model (see Section 6.3.3), this section will consider the delayed excitatory-inhibitory network version of the model. Similarly to the Wilson-Cowan model, the imposition of the row-sum normalisation on the connectivity matrix allows for the eigenvalue equation of the system to be reduced from a $14N$ by $14N$ determinant problem to N individual 14 by 14 determinant problems. This facilitates the much simpler computation of eigenvalues, as the numerical stability of the determinant calculation for large near-singular matrices can yield inaccurate results.

A major drawback of the Wilson-Cowan model is the need to incorporate a constant offset delay in order to excite a variety of eigenmodes. As

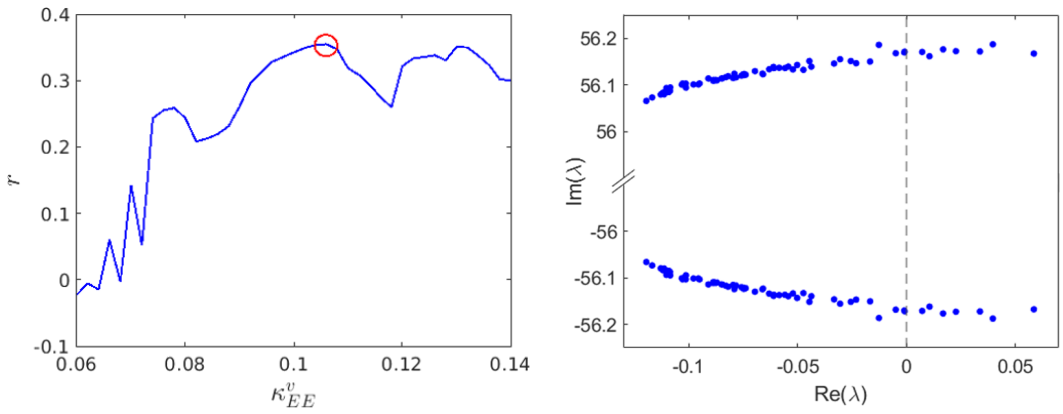
explained in Section 6.3.3, this offset delay does not have a natural physical interpretation, and is often only included in models to enable a richer array of dynamics [142]. In comparison to this, the next-generation model does not require the inclusion of this offset delay to generate intricate solutions. This is incredibly powerful as it pushes the model closer to goal of accurately recreating observed brain behaviour. For the remainder of this section, we consider a conduction velocity value of $2\text{m}\,\text{s}^{-1}$. This is slightly slower than the value of $10\text{m}\,\text{s}^{-1}$ used in the Wilson-Cowan results; however, it is still within the 1-$10\text{m}\,\text{s}^{-1}$ range that is commonly referred to in the literature [51, 45].

Parameter	Description	Value
τ_E	Excitatory time scale	0.3
τ_I	Inhibitory time scale	0.2
Δ_E	Excitatory degree of heterogeneity	0.9
Δ_I	Inhibitory degree of heterogeneity	0.9
κ_{EI}^v	Inhibitory-to-excitatory gap junction coupling	0.05
κ_{EI}^v	Excitatory-to-inhibitory gap junction coupling	0.05
κ_{EI}^v	Inhibitory-to-inhibitory gap junction coupling	0.1
η_0^E	Excitatory degree of excitability	0.2
η_0^I	Inhibitory degree of excitability	0.1
v_{syn}^{EE}	Excitatory-to-excitatory synaptic reversal potential	15.0
v_{syn}^{EI}	Inhibitory-to-excitatory synaptic reversal potential	-5.0
v_{syn}^{IE}	Excitatory-to-inhibitory synaptic reversal potential	15.0
v_{syn}^{II}	Inhibitory-to-inhibitory synaptic reversal potential	-5.0
α_{EE}	Excitatory-to-excitatory synaptic time scale	10.0
α_{EI}	Inhibitory-to-excitatory synaptic time scale	2.0
α_{IE}	Excitatory-to-inhibitory synaptic time scale	10.0
α_{II}	Inhibitory-to-inhibitory synaptic time scale	5.0
α_{ext}	Long-range synaptic time scale	10.0
κ_{EE}^s	Excitatory-to-excitatory synaptic coupling	0.5
κ_{EI}^s	Inhibitory-to-excitatory synaptic coupling	0.2
κ_{EI}^s	Excitatory-to-inhibitory synaptic coupling	0.4
κ_{EI}^s	Inhibitory-to-inhibitory synaptic coupling	0.4
κ_{ext}^s	Long-range synaptic coupling	0.2

Table 3: Next-generation network model parameters.

The other fixed parameters for this section are detailed in Table 3, and have been selected such that the solutions arising from a Hopf bifurcation oscillate with frequencies corresponding to the alpha band (8-12Hz). Here, the excitatory-to-excitatory gap junction coupling parameter, κ_{EE}^v,

is the chosen parameter of interest to be varied in order to push the spectrum unstable. In a similar fashion to the results presented in Section 6.3.3, we simulate the model for a variety of values of κ_{EE}^{v} (using the *DelaySparseRungeKutta32Solver* solver) and subsequently compute the AEC of each output, with the variable of interest being the excitatory firing rate, R_E. Each matrix is then compared by means of a Pearson correlation to the empirical FC data. Figure 37a shows the Pearson r value for each value of κ_{EE}^{v}. Indicated by the red circle is the maxima of this plot, highlighting that a value of $\kappa_{EE}^{v} = 0.106$ gives the highest correlation value of the AEC matrix of the output to the empirical FC. For this parameter set, the correlation value is $r = 0.3548$.



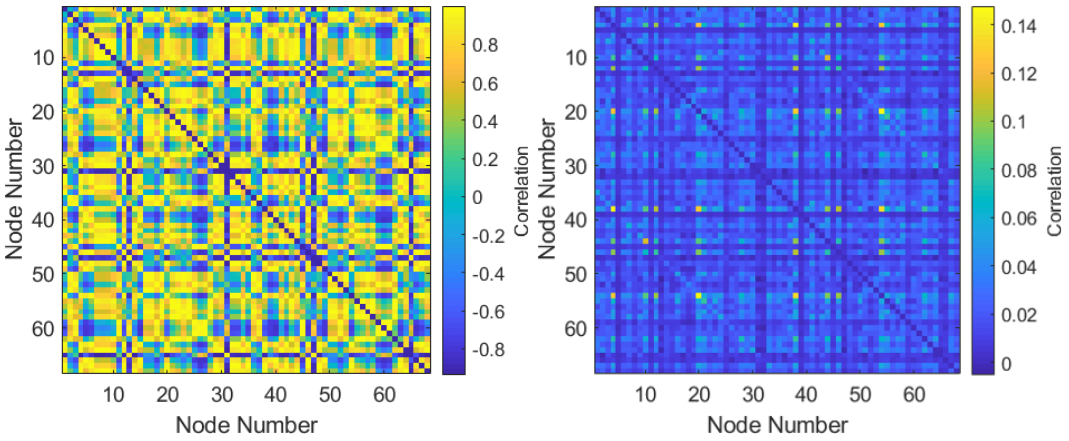
(a) Pearson correlation of the AEC of the model output compared to empirical FC.

(b) Spectrum of eigenvalues for the delayed next-generation network with $\kappa^v_{EE} = 0.106$, computed around the steady state, $\overline{x}$.

Figure 37: Figure (a) shows the Pearson correlation coefficient, r, for the comparison of the AEC of the simulated excitatory firing rate (R_E) signals against the empirical FC, computed for a range of values of the excitatory-excitatory gap junction coupling strength parameter κ^v_{EE}. The red circle indicates the maxima of the plot, occurring at $\kappa^v_{EE} = 0.106$, at which the simulation results yield the best fit to empirical FC. Figure (b) depicts the corresponding eigenvalue spectrum, computed around the steady state, $\overline{x}$, for this value of κ^v_{EE}.

In comparison to the Wilson-Cowan results, which yielded a maximum correlation value of $r = 0.2307$, this is a significant improvement. The eigenvalue spectrum of the system, for this parameter set, is depicted in Figure 37b. For illustration purposes, only the eigenvalues close to the stability border are shown. Following the same eigenmode ordering scheme detailed in Section 6.2.2, the seven unstable eigenmodes are led by the 68th eigenmode and tailed by the 62nd, with those in between appearing in descending ordered. There are far less unstable eigenmodes here than in the Wilson-Cowan results, which required fifteen unstable eigenmodes to give

the AEC output with the highest correlation value. The corresponding AEC matrix generated from the simulation output is shown below in Figure 38a, alongside the empirical FC for comparison.



(a) Simulated FC matrix.

(b) Empirical FC matrix.

Figure 38: Figure (a) shows the simulated FC matrix, generated via AEC, for the excitatory firing rate variable (R_E). Model parameters are given in Table 3, along with $\kappa^v_{EE} = 0.106$. On the right, figure (b) shows the empirical FC matrix, as introduced in Section 6.2.1. The Pearson r value that measures the correlation between the two matrices is $r = 0.3514$.

The Pearson correlation value for the next-generation results is significantly higher than that of the Wilson-Cowan results, which is backed up observationally by the next-generation simulated FC reflecting the features of the empirical FC far more than the Wilson-Cowan simulated FC. There are several key landmarks that appear in both the next-generation simulated FC and the empirical FC, mostly appearing in dark blue in the matrices shown in Figure 38. It is important to note that the Pearson correlation value is just one measure of similarity between the two matrices and is chosen here purely to aid in informing the conclusions taken from the results. Upon visual inspection, it can be possible that two matrices look more similar than a correlation measure indicates, or that the matrices do not look particularly alike even though a correlation measure argues a high level of similarity. It

is clear here that the next-generation model outperforms the Wilson-Cowan in both respects, as the resulting FC matrix has a higher correlation value and also visually appears to be far closer to the empirical FC matrix.

6.5 SUMMARY

This chapter has presented a selection of key results from the perspective of neural mass modelling. Initially, we introduced the different modalities of brain data (supplied by the HCP) and explored how SC and path length data can be incorporated into models of neural activity in an attempt to replicate FC data. Following on from this, we highlighted methods used by Tewarie et al. [164] to break down the eigenstructure of an SC matrix and apply a fitting algorithm to determine which eigenmodes best reflect FC data. Although these methods are useful in highlighting eigenmodes that could lead to better FC fits, in practice, it is usually difficult to locate parameter regimes that excite just the desired eigenmodes in a dynamic model, while also taking into account aspects such as the frequency of the resulting oscillations and the biological plausibility of the selected parameter values.

Section 6.3 proceeds to introduce and discuss the Wilson-Cowan neural mass network model. Beginning with the model description, the section then moves on to presenting the linear stability analysis of the network steady state for the model. This uses key results from Section 3.2, showing how the eigenvalue equation can be reduced to N independent 2×2 determinant problems, each in terms of a distinct eigenvalue of the SC matrix. This reduction follows in a similar fashion under the incorporation of delays in the system, except each 2×2 problem becomes dependent on a distinct pair of left and right eigenvectors of the SC matrix, and is transcendental in the eigenvalue, λ. Making use of this linear stability analysis, we show that in the absence of delays, the eigenmodes

can only be excited in descending consecutive order. To explore a wider range of results, we show that it is necessary to consider the model with the incorporation of axonal delays. However, a key drawback of the Wilson-Cowan model is that it requires a constant offset delay to be added to the space-dependent delays in order to excite a different eigenmode pattern to the non-delay case. This offset delay, as discussed in Section 2.3.2, is not as biologically realistic as the space-dependent delays, and is often only included in models as it can yield richer dynamics, similar to those seen in simulations of large-scale spiking networks [142]. To illustrate how this offset delay affects the eigenvalue spectrum, we plot the spectrum for a range of values of the offset delay. It becomes evident that as the offset delay is increased, the spectrum rotates so that it can be excited from the tail end, in ascending order, instead of the descending order of the non-delay case. Selecting an appropriate value that gives the rotated spectrum, we then proceed to simulate the model for a range of values of the excitatory drive parameter, P, through the use of the solvers in the NFESOLVE library. For each simulation, we use AEC to generate a FC matrix from the excitatory firing rate variable (E) signals, and subsequently compare it to the empirical FC data by means of a Pearson correlation. As the aim is to best replicate the empirical FC matrix, we take the highest correlation value, for the range of P considered, and plot a visualisation of the simulated FC matrix next to the empirical FC matrix. The resulting correlation value is determined to be $r = 0.2307$; however, upon visual inspection of the two matrices shows, it is clear that they do not share a lot of the structural elements present in the empirical FC matrix.

With the recent emergence of next-generation models of neural activity, comes the question of whether the Wilson-Cowan model can be outperformed in a similar setting to that presented above. Section 6.4 begins by introducing the model developed by Coombes and Byrne [36]. This is based on techniques previously explored by Montbrió et al. [124].

The derivation begins with a network of N QIF neurons that incorporates gap junction coupling, synaptic coupling, and synaptic reversal potentials. To make the bridge from microscopic to macroscopic dynamics, the thermodynamic limit ($N \to \infty$) of the network is considered. By means of a Lorentzian ansatz, the model is finally reduced to a system of four first order equations (due to the choice of α-function response yielding a second order linear differential operator), with variables representing the average firing rate, voltage, and global conductance of the population of neurons. We then showed how this model can be extended to two sub-populations (excitatory and inhibitory), followed by the construction of a network of populations. An exploration of the linear stability analysis was then undertaken, utilising the same technique as with the Wilson-Cowan model to reduce the eigenvalue equation from a $14N \times 14N$ determinant problem to N individual 14×14 determinant problems. The incorporation of delays into the model was also accounted for, making it clear that for the next-generation model it was not necessary to include a constant offset delay to excite a variety of different eigenmodes. In order to compare the next-generation model to the Wilson-Cowan model, it was necessary to locate a parameter regime which led to the simulation output exhibiting oscillations occurring with similar frequency. Fixing all parameters except for the excitatory-to-excitatory gap junction coupling parameter, κ_{EE}^{v}, we then simulated the model for a range of κ_{EE}^{v}. Once again, taking the AEC of the excitatory firing rate parameter (R_E), FC matrices were generated for each value of κ_{EE}^{v} and compared to the empirical FC by means of a Pearson correlation in order to find the parameter regime that yielded solutions that could best describe the empirical FC. The AEC FC matrix that gave the best fit had a correlation value of 0.3514, which was significantly higher than that of the Wilson-Cowan model. Visual inspection also allowed the viewer to note the much more defined structural elements of the AEC FC matrix, especially reflective of those seen in the empirical FC.

The main observations and conclusions that can be drawn from this chapter are that although the Wilson-Cowan model has been a cornerstone in modelling and simulating neural activity, it clearly has its drawbacks. The need for the inclusion of the biologically unrealistic constant offset delay to match data at the network level, is a key characteristic in favour of the next-generation model for this study. This is not to say that the Wilson-Cowan model is inferior to the next-generation model; however, in the context presented in this chapter, the results show that the next-generation model can generate simulation output that is more reflective of empirical FC data, and due to the large number of physiologically meaningful parameters available to explore and the number of equations present in the model, it has more scope to be used in different settings and to generate a wider variety of solutions. Although the next-generation model is marginally more computationally expensive to simulate than the Wilson-Cowan model, it is more firmly rooted in biological reality and can evidently generate desirable and relevant results.

There is also the question of whether it is realistic to consider only solutions arising from instabilities of a steady state. The row-sum normalisation of the SC matrix facilitates the existence of a homogeneous steady state solution and allows for the simple computation of the system's eigenvalues via the linear stability analysis described in Section 3.2, which makes the location of a dynamic model solution a straightforward task. However, the disadvantage of this approach is that only considering solutions near to bifurcation may restrict the scope of the results that are possible. Conversely, there have been several studies around the brain and how it is theorised to operate at a critical state [13, 137, 182, 14]. These explore the relationship between criticality and models of neural networks. Deco et al. [46, 47] specifically study the role criticality plays in the emergence of resting state functional connectivity patterns produced by a network model, and conclude that the brain appears to operate close to a critical state when at rest.

7

NEURAL FIELDS ON REALISTIC CORTICAL DOMAINS

7.1 INTRODUCTION

This chapter will showcase the capabilities of the NFESOLVE library by utilising it to solve a variety of problems posed an a realistic cortical domain. We explore three different models of neural activity with the inclusion of space-dependent axonal delays. These models are the standard neural field equation, the neural field equation with linear adaptation, and the next-generation neural field model (as introduced for a network of neural masses in Chapter 6). In each instance, we utilise data supplied by the Human Connectome Project (HCP) to provide a simulation context closer to reality. We present this data in Section 7.2, before moving on to explore the specified models of neural activity and some example solutions that each model yields. Supplementary material, such as movies of the presented dynamic solutions, can be found at `https://github.com/sammyjp/Thesis-Supplementary-Material`.

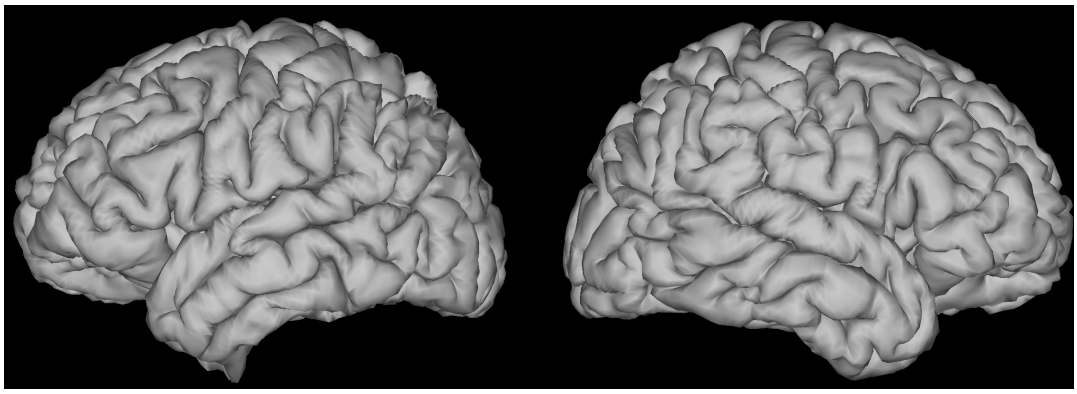
7.2 DATA

Similarly to the networks of neural masses explored in Chapter 6, this chapter also aims to incorporate real brain data into simulations of neural field models. These models are posed on a continuum domain, which for the

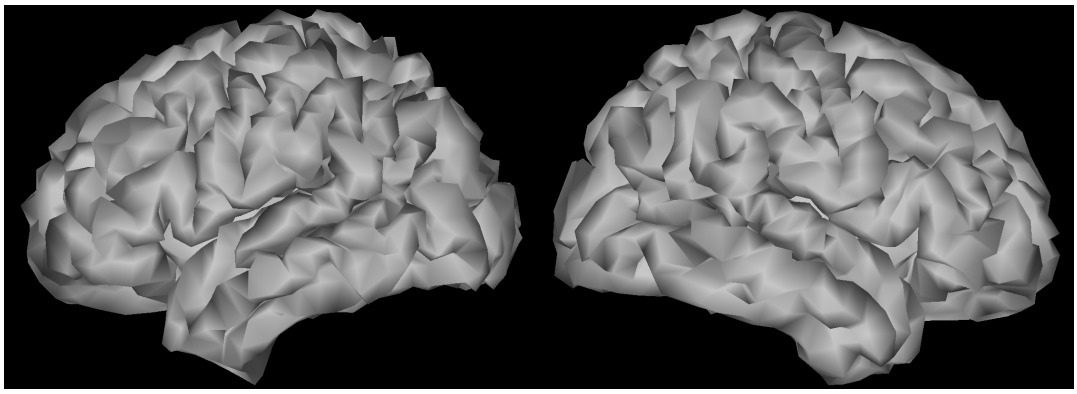
scope of this body of work we select to be a cortical surface. The data utilised in this chapter is once again supplied by the HCP. Before exploring the specific models that will incorporate the data, this section will first detail the chosen cortical mesh, along with the corresponding structural connectivity and path length data. All data presented here is based on average measurements taken from a single subject.

7.2.1 *Cortical Mesh*

To numerically solve continuum models of cortical activity, the spatial domain must be dicretised into a set of unique nodes (see Section 3.3 for more details). The original mesh data supplied by the HCP is incredibly detailed and contains $32,492$ nodes per hemisphere. In practice, it is extremely computationally expensive to simulate a system with such a large number of mesh points, especially with the incorporation of delays. The HCP have developed a powerful tool named Connectome Workbench [119] that allows the user to manipulate, resample, and visualise HCP data. One of Connectome Workbench's key features is the ability to down-sample data to a specified degree while preserving the sample space, allowing for the corresponding connectivity and path length data to also be down-sampled accordingly in the same sample space. When simulating, in order to compute solutions in a viable time frame, ideally a balance should be struck between computational efficiency and the degree of granularity that is present in the mesh. We wish to preserve the geometric details of the brain, such as the gyri and sulci, in order to push the simulations as close towards a realistic setting as possible, but it is not necessarily required to use the full resolution mesh to accurately represent such features.



(a) Surface mesh of the left and right hemispheres of the cortex with 32,492 nodes per hemisphere.



(b) Surface mesh of the left and right hemispheres of the cortex with 2,892 nodes per hemisphere.

Figure 39: The contrast between the original HCP cortical mesh with a large number of nodes and a down-sampled mesh with fewer nodes. Visualisation generated using Connectome Workbench.

Figure 39 shows the original high-resolution mesh compared to a down-sampled version containing only 2,892 nodes per hemisphere. This level of granularity has been selected due to the computational expense of simulating models with space-dependent delays on higher resolution meshes. It is clear that although some of the detail is lost, the main mesh features are preserved. With the provision of higher computing power, the number of mesh nodes could be increased to achieve a more accurate approximation to the solution. However, as the computational complexity of the delay problem scales in proportion to the square of the number of nodes in the mesh, it

soon becomes a challenging computational problem to overcome. Although the full lower-resolution mesh contains 2,892 nodes per hemisphere, the medial wall is not included in simulations, thereby reducing the node count to 2,652 per hemisphere.

7.2.2 *Structural Connectivity*

As introduced in Sections 2.4.1 and 6.2.1, a structural connectome describes the strength of the long-range cortical connections. This is an important data set that can easily be incorporated into neural field models to facilitate the modelling of non-local interactions. In this chapter, we consider the structural connectome from a single subject. This is visualised below in Figure 40, with blue representing weak connections and yellow depicting strong connections. The data has been normalised to take values on the range $[0, 1]$.

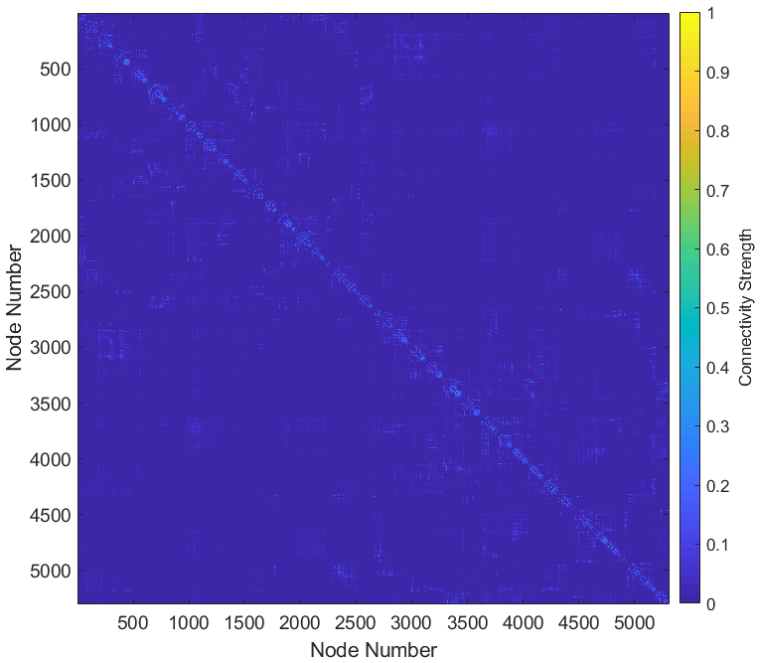


Figure 40: Structural connectome data from a single HCP subject, normalised to $[0, 1]$.

Similarly to the data used in Chapter 6, this connectome is extremely sparse, with very few high-strength connections. As explained in Section 6.2.1, this can be down to the processing that is undertaken at the acquisition stage. To reveal more structure in the connectivity patterns, the data can be log transformed and re-normalised [159]. Due to the intense computational requirements of solving neural field models with space-dependent delays, the computational load can be relieved somewhat by thresholding the structural connectivity matrix to cull values close to zero. After the data has been logged and re-normalised, all values less than a selected threshold are set to zero. For the purposes of this chapter, the threshold value is chosen to be 0.01, with less than one percent of the values in the matrix being smaller than that value. This helps to maintain the core structural pattern of the matrix, but allows the delay solvers to perform more efficiently by skipping computations for delay state values that would ultimately end up being multiplied by zero. As seen in Figure 41, a much more intricate network of connections is revealed when the log transform and thresholding is applied.

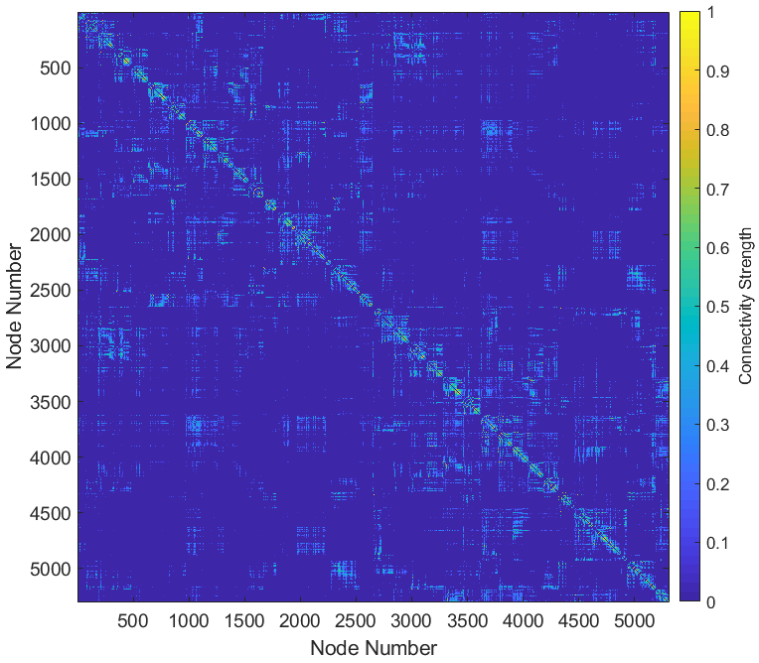


Figure 41: Log transformed connectome data, followed by normalisation to $[0, 1]$, and thresholding such that values less than 0.01 are set to zero.

An important point to note is that the data shown in Figure 41 depicts purely excitatory connections. This is because the tractography algorithms that map the pathways only determine the absolute strength of a connection, not the resulting excitatory/inhibitory impact. For more information on the data acquisition techniques, please refer to Section 2.4.

7.2.3 *Path Lengths*

Along with the structural connectivity data, the HCP also supply the corresponding path length data for the connections. This data is particularly relevant to models of neural activity when it comes to computing any terms that are space-dependent. In the absence of specfic path length data, typically a Euclidean or geodesic distance is used to approximate the lengths of the paths between nodes [105, 177]. These path lengths play a key part in models that incorporate delays, as it is most commonly assumed that ax-

onal delays are space-dependent, with a directly proportional relationship between distance and delay.

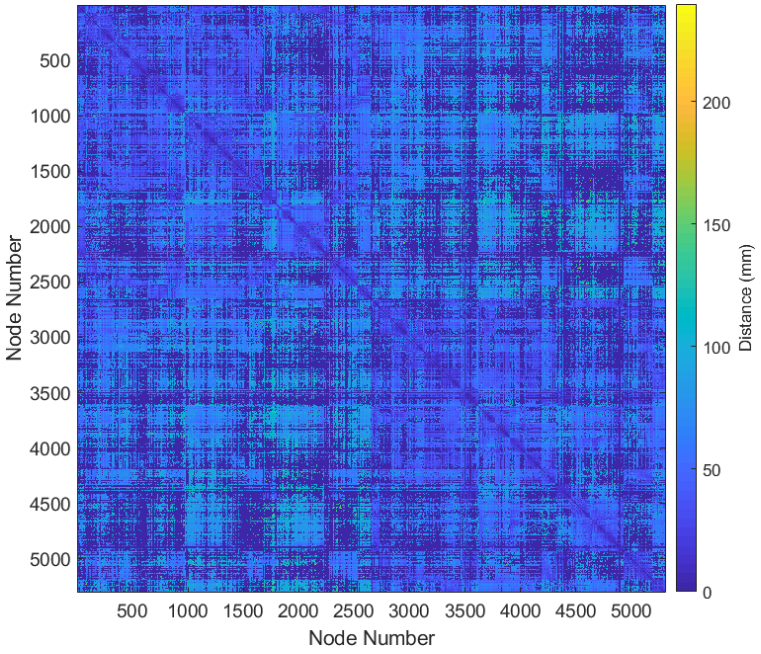


Figure 42: Path length data from a single HCP subject, showing the distance (in millimetres) between each node of the cortical mesh shown in Figure 39b.

Shown above, in Figure 42, is a visualisation of the path length data that will be utilised in the simulations that follow in this chapter. Regions in yellow depict long-distance connections and regions in blue show short-range connections, with a distance of zero representing non-existent connections.

7.3 STANDARD NEURAL FIELD EQUATION

The first model that we focus on here is the standard NFE. This model has been studied by the mathematical neuroscience community for several decades, and forms the basis of a number of more advanced models. However, there has been comparatively little exploration of the outcomes yielded by the model when it is posed on a realistic cortical domain and incorporates real brain data. In this section, we explore the formulation of the

model in the presence of space-dependent delays, and present some example solutions that arise when the model is considered on a cortex.

7.3.1 *The Model*

As seen throughout this thesis, the equation for the standard neural field model is given by

$$\tau \frac{\partial}{\partial t} u(\boldsymbol{x}, t) = -u(\boldsymbol{x}, t) + \int_{\Omega} w(\boldsymbol{x}, \boldsymbol{x}') f\left(u(\boldsymbol{x}', t)\right) \mathrm{d}\boldsymbol{x}', \tag{209}$$

where u, $\boldsymbol{x}$, t, and Ω, represent brain activity, the spatial coordinates, time, and the entire spatial domain, respectively. The connectivity kernel that facilitates the non-local interactions is given by w, and the firing rate function, f, is the standard sigmoidal response function used throughout this thesis (with steepness parameter, μ, and threshold parameter, θ), as originally defined in Section 2.3.2. A temporal scaling parameter, τ, is also included.
In order to adapt the model to more realistically describe observed phenomena, space-dependent delays can be incorporated into the non-local interactions. The relevance and significance of delays in models of neural activity has been introduced previously in Section 2.3.2, along with the exploration and simulation of a variety of neural models with delays in Chapters 5 and 6.
The delayed standard NFE takes the form

$$\tau \frac{\partial}{\partial t} u(\boldsymbol{x}, t) = -u(\boldsymbol{x}, t) + \int_{\Omega} w(\boldsymbol{x}, \boldsymbol{x}') f\left(u(\boldsymbol{x}', t - s(\boldsymbol{x}, \boldsymbol{x}'))\right) \mathrm{d}\boldsymbol{x}', \tag{210}$$

where the space-dependent delay term is represented by $s(\boldsymbol{x}, \boldsymbol{x}')$. As seen in previous chapters, this term often takes the form

$$s(\boldsymbol{x}, \boldsymbol{x}') = \tau_0 + \frac{\|\boldsymbol{x} - \boldsymbol{x}'\|}{v}, \tag{211}$$

where τ_0 is a constant offset delay, $\|\cdot\|$ is some measure of distance, and v is the axonal conduction velocity. Although not as physically relevant (as discussed in Section 2.3.2, the offset delay term, τ_0, has been shown to aid

in yielding dynamic solutions similar to those seen in simulations of large-scale spiking networks [142], hence we choose to include it as a tunable parameter. It is also extremely helpful to include as the maximum step size of the chosen delay solver is restricted to the minimum delay value due to the interpolation step in the algorithm (see Appendix A.2.3 for details on continuous output methods for DDEs). This means that as τ_0 increases, so does the maximum possible step size that the solver can adapt to. This can potentially lead to quicker solution times, depending on the smoothness of the solution. For the case of the fast oscillatory solutions that we seek, there may not always be noticeable changes in computation time as a smaller step size may be required to account for the higher frequency solution; however this is also dependent on just how small the original minimum delay value actually is.

To numerically solve Eq. (210), we first employ a Nyström discretisation (as introduced in Section 3.3) to discretise the spatial component. As the mesh, structural connectivity data, and path length data are already at a pre-determined resolution, we select the integration nodes to coincide with the existing mesh points. Introducing new integration nodes would require the data to be interpolated, thereby introducing another instance of numerical error into the system, so it makes sense to consider the existing mesh points as the integrations nodes. Using the quadrature library implemented in the NFESOLVE library (see Section 4.3), a set of quadrature weights can easily be generated for a vertex quadrature rule on the cortical mesh. Although not as accurate as a higher order quadrature rule (such as Gaussian quadrature), this specific vertex rule is analogous to the trapezoid rule in one dimensional space, which is commonly used to numerically solve NFEs [120]. The spatially discretised form of Eq. (210) is written as

$$\tau \dot{u}_i = -u_i + \sum_{j=1}^{N} w_{ij} f\left(u_j\left(t - s_{ij}\right)\right) \sigma_j, \tag{212}$$

for $i = 1, \ldots, N$, where N is the total number of mesh points, w_{ij} is the structural connection strength between the ith and jth nodes, σ_j is the quadrature

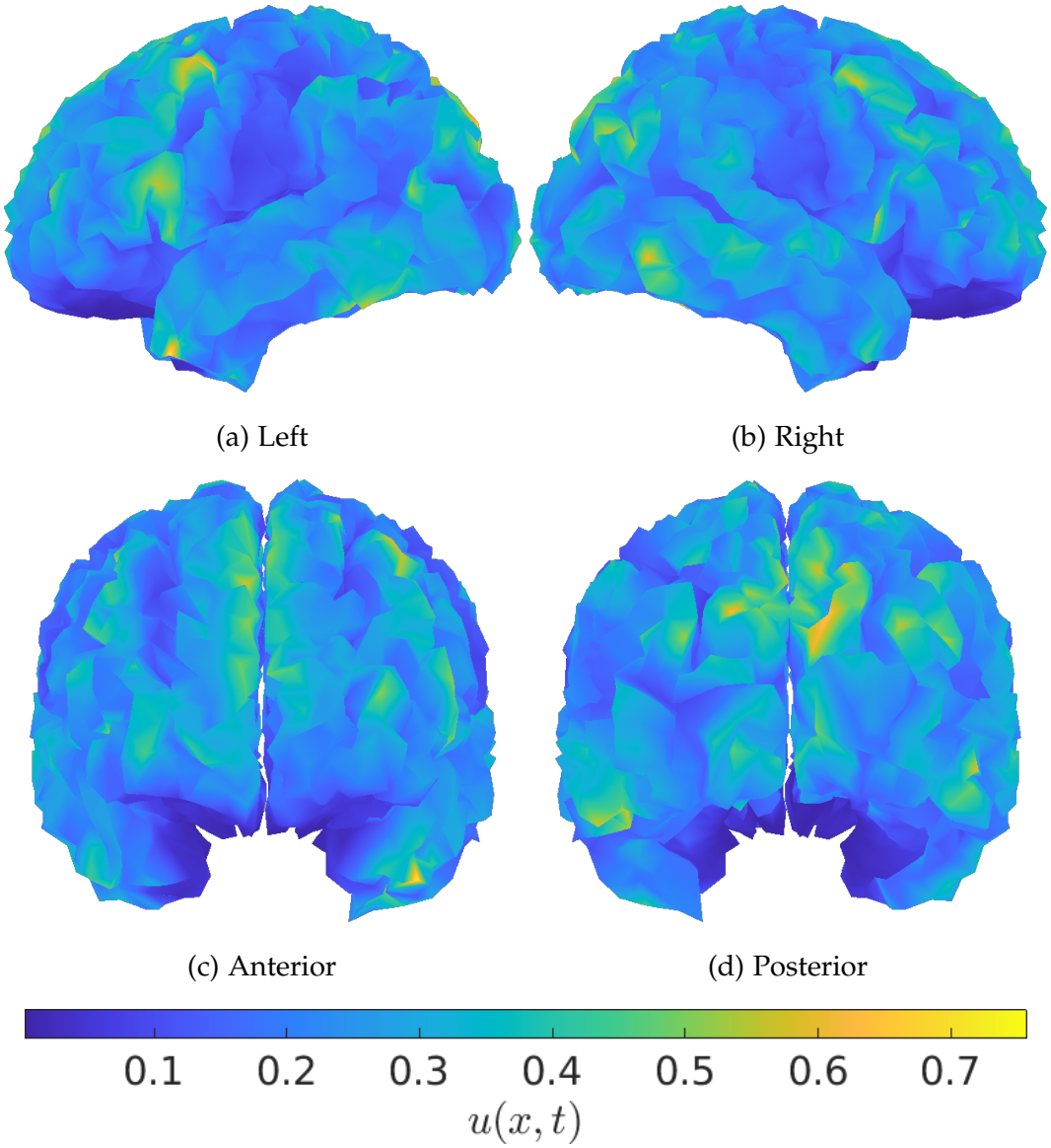
weight corresponding to the jth node, and $s_{ij} = \tau_0 + {}^{D_{ij}}/_{v}$, with D_{ij} representing some discrete distance between nodes i and j, corresponding to the measure given by $\|\cdot\|$. For the purposes of our simulations, we select these distances to be the path length data provided by the HCP, as presented in Section 7.2.3. In all simulations in this chapter, we further apply a normalisation to the structural connectivity data such that

$$\max_i \sum_j^N w_{ij}\sigma_j = 1, \tag{213}$$

for $i = 1, \ldots, N$, in order to ensure that the magnitude of the integral approximation is controlled to appropriately.

7.3.2 *Examples*

The system given by Eq. (212) can be solved in an efficient manner by the DDE solvers in the NFESOLVE library. For the remainder of this chapter, we make use of the adaptive-step solver, *DelaySparseRungeKutta32Solver*, to evolve all of the presented solutions. This solver is chosen to allow results to be computed in an efficient manner while maintaining an appropriate level of accuracy. As explained in Section 4.6, this solver relies on the sparsity pattern of the delay state matrix, Z. This is very simple to compute for the space-dependent delay set up used here. The choice to use an adaptive-step solver is also an important decision, as an adaptive-step solver can often be more useful than a fixed-step solver when solving delay equations. When dealing with a large number of delays the computational intensity can be extremely high, meaning that any potential saving gleaned by adapting the step size during smoother areas of the solution is often welcome.



(a) Left (b) Right

(c) Anterior (d) Posterior

Figure 43: Four perspectives of an example steady state solution to the standard NFE posed on a cortical domain. Parameter values: $\mu = 20$, $\theta = 0.1$, $\tau = 1$, $\tau_0 = 0.01$, $v = 10000$.

For all simulations in this section, the temporal scaling parameter, τ, is fixed to a value of $\tau = 1$. For the delay parameters, as the path length data is given in millimetres, we select the axonal propagation speed to have a value of $v = 10,000\text{mm}\,\text{s}^{-1}$, as we wish the temporal unit of the solution to be seconds. As explained in Section 2.3.2, the velocities of action potentials along axons averages around $10\text{m}\,\text{s}^{-1}$, which is why we have selected this value of v. In order to aid the efficiency of the temporal evolution algorithm, we opt to also introduce a fixed delay of $\tau_0 = 0.01\text{s}$ (10ms), so that the maximum step size of the solver is not too small to generate solutions in an acceptable time frame. This is small enough that it should not have a large impact on the solution.

Figure 43 depicts an example steady state solution to Eq. (210). In this example, the firing rate steepness parameter is selected to have a value of $\mu = 20$, and the firing rate threshold parameter is chosen to be $\theta = 0.1$. This particular stationary state appears to form a pattern depicting global mid-level activity with spots of high and low activity positioned around the cortex. Through simulation across a wide range of parameter values, a key observation that we gleaned is that the majority of the steady states yielded follow an extremely similar patterned structure, although often across various degrees of magnitude. There are, however, some states that exhibit a differing stable pattern from that seen in Figure 43. An example of this can be seen in Figure 44, where the majority of the cortex features low level activity, but the anterior end of the cortex is illuminated with activity. The firing rate parameter values used for this simulation are $\mu = 20$ and $\theta = 0.15381$.

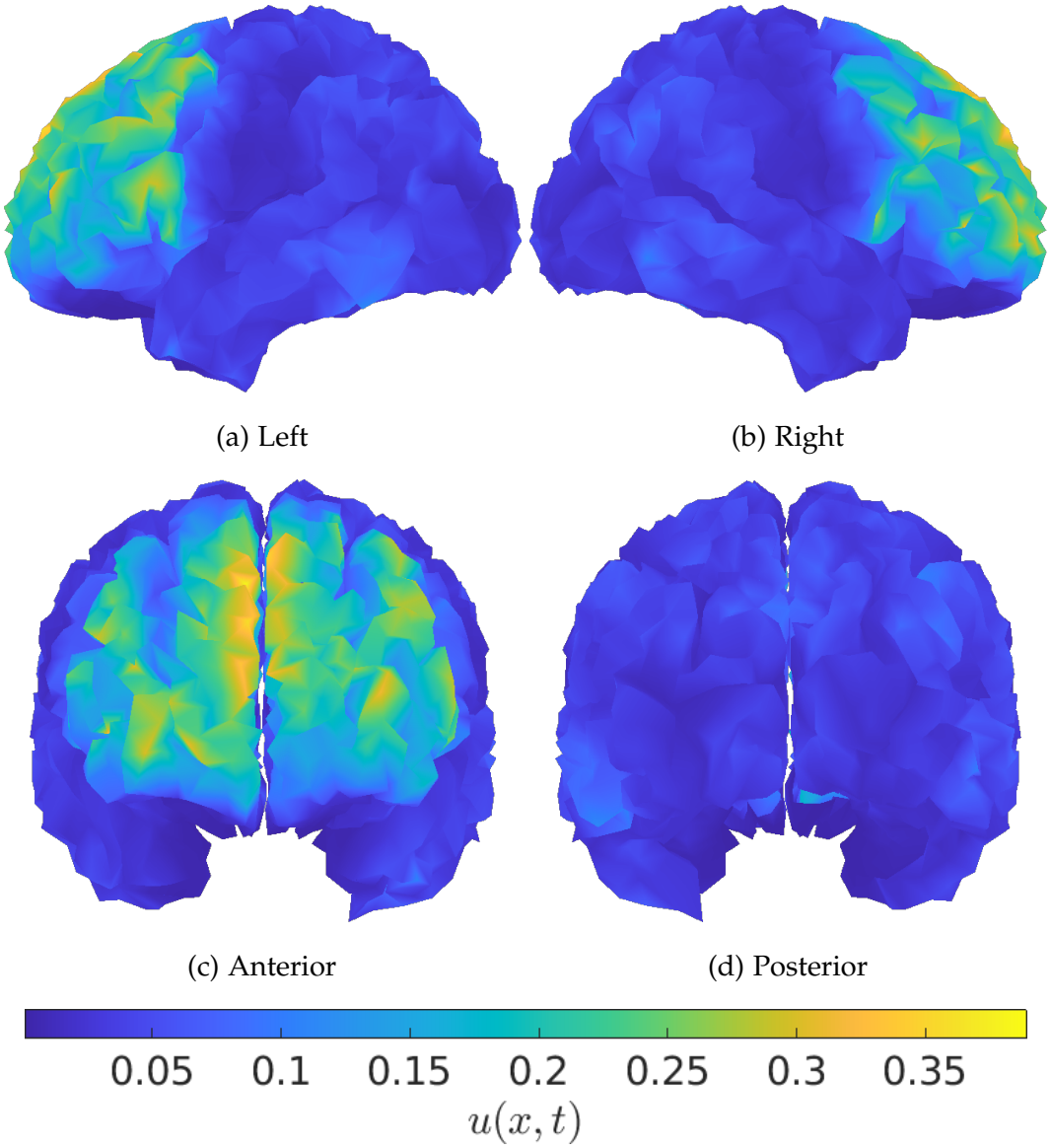


Figure 44: Four perspectives of an example steady state solution to the standard NFE posed on a cortical domain. Parameter values: $\mu = 20$, $\theta = 0.15381$, $\tau = 1$, $\tau_0 = 0.01$, $v = 10000$.

Although there are a selection of interesting stationary patterns generated by the delayed NFE, ultimately it is severely limited when it comes to recreating observed brain activity data. Due to limitations around the production of this work, it has proved difficult to locate interesting dynamic patterns for the delayed NFE with the data available. However, as reflected in the 'Future Work' section of this thesis (Section 8.2), there is evidently

more that can be done around the exploration of the model to yield more dynamic results. In the meantime, in order to produce output more reflective of real-life brain activity patterns, there are other, more developed models that can be considered.

7.4 NEURAL FIELD WITH LINEAR ADAPTATION

As true brain activity patterns are not static, it is preferable to explore models of neural activity that allow for dynamic solutions to arise more readily. One such model takes the standard NFE and incorporates an additional variable in the form of linear adaptation [135]. As introduced in Section 2.3.3, this can be thought of as implementing an intrinsic negative feedback mechanism that neurons possess, whereby they actively work to decrease the activity levels after periods of high activity.

7.4.1 *The Model*

Firstly, we consider the model in the absence of delays. Building upon the standard NFE (Eq. (209)), the model takes the form

$$\tau \frac{\partial}{\partial t} u(\boldsymbol{x}, t) = -u(\boldsymbol{x}, t) - g a(\boldsymbol{x}, t) + \int_{\Omega} w(\boldsymbol{x}, \boldsymbol{x}') f\left(u\left(\boldsymbol{x}', t\right)\right) \mathrm{d}\boldsymbol{x}', \tag{214}$$

$$\frac{1}{\alpha} \frac{\partial}{\partial t} a(\boldsymbol{x}, t) = u(\boldsymbol{x}, t) - a(\boldsymbol{x}, t), \tag{215}$$

where the additional variable, a, is a dummy variable that facilitates the negative feedback. This is also referred to as 'adaptation', as the neurons are adapting to constant stimulus by decreasing their activity levels. Including this additional variable naturally allows for oscillatory instabilities to occur in the non-delay case, unlike the standard NFE, due the model now being multi-dimensional. The model parameters represent the same things as that of the standard NFE, with the addition of two new parameters, g and α, representing the coupling strength of the feedback and the rate at which

the adaptation occurs, respectively. When the coupling strength, g, is equal to zero, the model is equivalent to the standard NFE. Extending the model to incorporate delays is analogous to that of the standard NFE, with the non-local interaction term in Eq. (216) being modified to give the equation

$$\tau \frac{\partial}{\partial t} u(\boldsymbol{x}, t) = -u(\boldsymbol{x}, t) - g a(\boldsymbol{x}, t) + \int_{\Omega} w(\boldsymbol{x}, \boldsymbol{x}') f\left(u\left(\boldsymbol{x}', t - s(\boldsymbol{x}, \boldsymbol{x}')\right)\right) \mathrm{d}\boldsymbol{x}', \quad (216)$$

where, as seen previously, the delay term is chosen to take the form

$$s(\boldsymbol{x}, \boldsymbol{x}') = \tau_0 + \frac{\|\boldsymbol{x} - \boldsymbol{x}'\|}{v}. \quad (217)$$

In order to numerically solve this system of equations, the techniques introduced in Section 3.3.4 can be employed to spatially discretise the system, giving a system of N DDEs of the form

$$\tau \dot{u}_i = -u_i - g a_i + \sum_{j=1}^{N} w_{ij} f\left(u_j\left(t - s_{ij}\right)\right) \sigma_j, \quad (218)$$

$$\frac{1}{\alpha} \dot{a}_i = u_i - a_i, \quad (219)$$

for $i = 1, \ldots, N$. Once again, w_{ij} is the structural connectivity strength between node i and j, σ_j is the quadrature weight corresponding to the jth spatial node, and $s_{ij} = \tau_0 + {}^{D_{ij}}/_{v}$, for some discrete distance, D_{ij}, between the ith and jth nodes. In the following simulations, these distances are again selected to be the anatomical path lengths supplied by the HCP, as introduced in Section 7.2.3.

7.4.2 *Examples*

Similarly to Section 7.3.2, here we present some example solutions to the NFE with linear adaptation. As the model is equivalent to the standard NFE when the coupling strength, g, is set to zero, we use the stationary state depicted in Figure 43 as a basis for locating a dynamic solution.

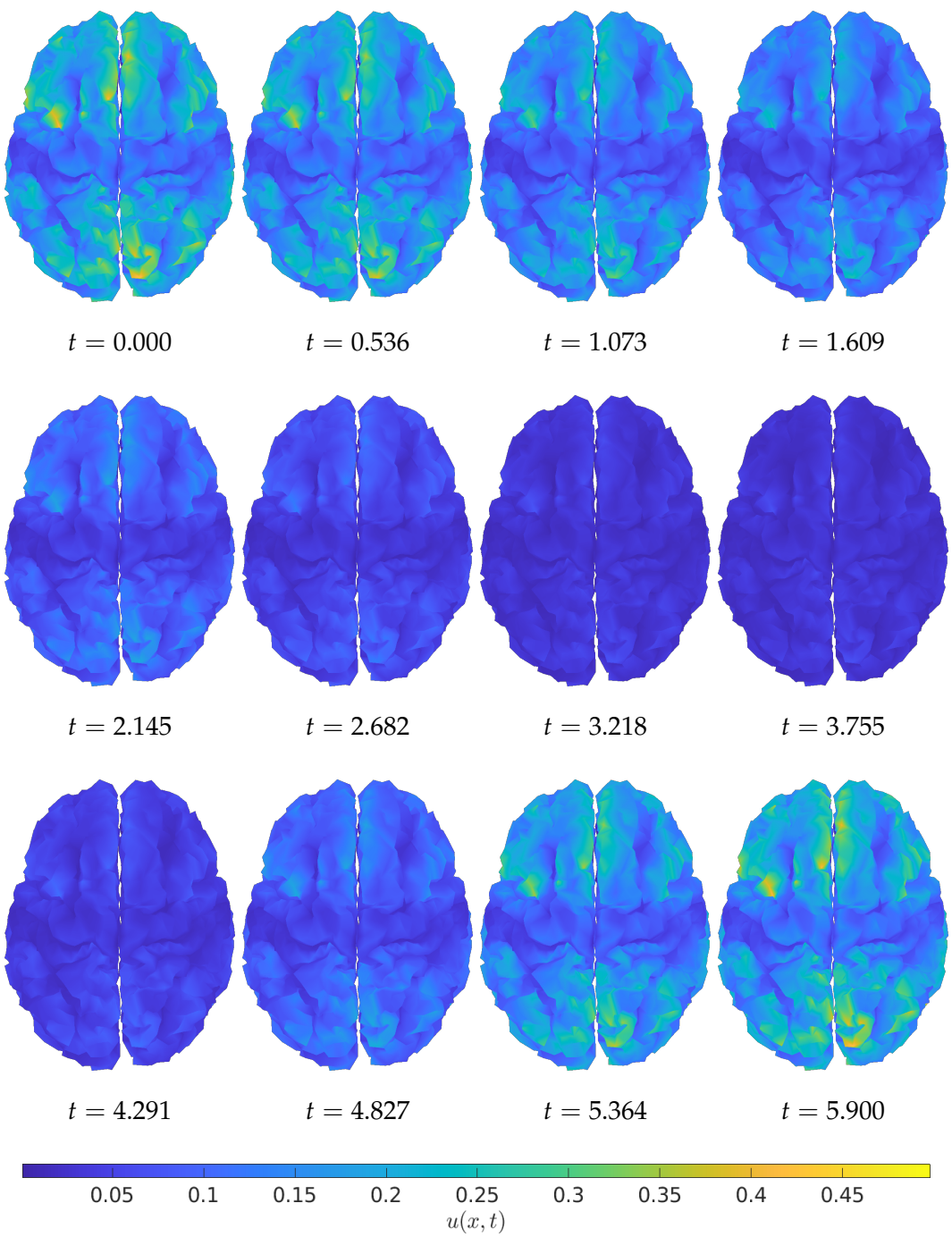


Figure 45: Dorsal view of one period of an oscillatory solution to the NFE with linear adaptation, lasting approximately 5.9 seconds, arising from a Hopf bifurcation of the stationary state. The state variable depicted here is the activity variable, $u(\boldsymbol{x}, t)$, and the period is divided into 12 equally spaced frames with times given to three decimal places. Parameter values: $\mu = 20$, $\theta = 0.1$, $\tau = 0.23$, $\alpha = 1$, $g = 0.6$, $\tau_0 = 0.01$, $v = 10,000$.

Fixing the firing rate parameters to $\mu = 20$ and $\theta = 0.1$, the coupling strength can slowly be increased to observe the emergence of other solutions, with the temporal scaling parameters, τ and α, being tuned to determine the frequency of the oscillations. The delay parameters are fixed to $v = 10,000\text{mm}\,\text{s}^{-1}$ and $\tau_0 = 0.01\text{s}$, as explained in Section 7.3.2.

One period of an oscillatory solution is illustrated over 12 frames in Figure 45, with the variable of interest being the activity variable, $u(\boldsymbol{x}, t)$. For this simulation, we select a coupling strength value of $g = 0.6$, and temporal scaling values of $\tau = 0.23$ and $\alpha = 1$. This yields an oscillatory solution with a period of approximately 5.9 seconds. The pattern of activity in this simulation remains visually similar to the steady state; however, the oscillatory nature of the solution causes a gradual decrease in activity globally across the cortex. As the oscillation reaches its trough, the definition in the starting pattern is lost as the magnitude of the range of the solution at that point becomes much smaller than to start with. The pattern then proceeds to rise back to the peak of the oscillation at a slightly faster pace than which it descended. Note that the area, best described as a horizontal band across the centre of the cortex, oscillates with a much smaller amplitude than the anterior and posterior regions.

Another example is depicted in a similar fashion in Figure 46. In this example, the firing rate steepness parameter is increased to a value of $\mu = 40$, and the coupling strength parameter is reduced slightly to a value of $g = 0.5$. The resulting pattern, still a global oscillation, appears to exhibit the opposite behaviour of that observed in the first example. The regions further away from the central lateral band (closer to the anterior and posterior ends of the cortex) oscillate with a much smaller amplitude, such that the effect is barely visible in the simulations. Whereas, the middle region much more noticeably fluctuates to a lower magnitude of activity.

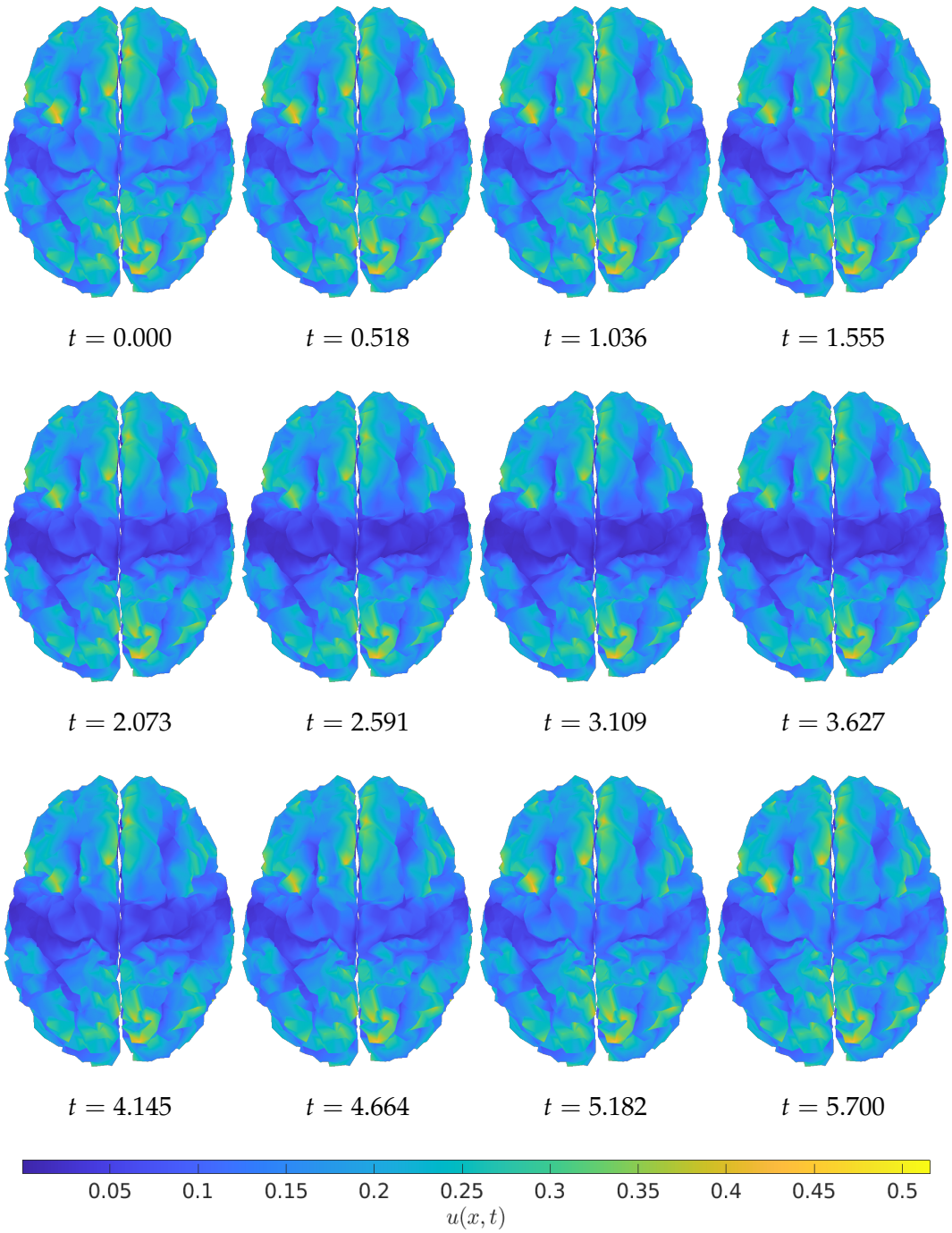


Figure 46: Dorsal view of one period of an oscillatory solution to the NFE with linear adaptation, lasting approximately 5.7 seconds, arising from a Hopf bifurcation of the stationary state. The state variable depicted here is the activity variable, $u(\boldsymbol{x}, t)$, and the period is divided into 12 equally spaced frames with times given to three decimal places. Parameter values: $\mu = 40$, $\theta = 0.1$, $\tau = 0.2$, $\alpha = 1$, $g = 0.5$, $\tau_0 = 0.01$, $v = 10,000$.

Although these two example solutions exhibit oscillatory behaviour, in both cases the oscillations occur globally. Similarly to the standard delayed NFE, the model does theoretically provide the ability to produce intricate and heterogeneous dynamic solutions; however, due to the computational difficulty of carrying out numerical stability analyses of large-scale delay equations, these more advanced solutions are challenging to locate. The generation of the solutions depicted was largely down to trial and error, aided by the swift computations of the NFESOLVE library. The 'Future Work' section of this thesis (Section 8.2) describes the potential avenues that could be pursued to explore the wider variety of solutions that this model has to offer.

7.5 NEXT-GENERATION NEURAL FIELD MODEL

The models discussed previously in this chapter clearly have their limitations when it comes to recreating patterns of neural activity. As introduced in Chapter 6, Coombes and Byrne [36, 25, 26] have recently developed a next-generation model that has a much stronger foundation in observed biological processes than models such as the standard NFE. In Section 6.4, we discussed the derivation of the model in detail and explored a selection of results in the setting of a neural mass network. Here, we extend the work of Section 6.4 to consider the neural field version of the model.

7.5.1 *The Model*

Although the field model was originally introduced in [25, 26], the version that we choose to study in this chapter is closer to that presented in [29]. The model has been evolved to now incorporate gap junction coupling, synaptic

coupling, and synaptic reversal potentials (all of which are explained in Section 6.4.1). The field version is given by the equations

$$\tau \frac{\partial R}{\partial t} = 2RV + \frac{\Delta}{\pi\tau} - R\left(g + \kappa^v\right), \tag{220}$$

$$\tau \frac{\partial V}{\partial t} = \eta_0 - \pi^2\tau^2 R^2 + V^2 + g\left(v_{\text{syn}} - V\right), \tag{221}$$

$$Qg = \kappa^s \Psi, \tag{222}$$

where $(R, V, g) = \left(R(\boldsymbol{x}, t), V(\boldsymbol{x}, t), g(\boldsymbol{x}, t)\right)$ represent the firing rate, average membrane potential, and synaptic activity, respectively. The linear differential operator, Q, is given by $Q = \left(1 + \alpha^{-1}\partial/\partial t\right)^2$, and Ψ is the spatial convolution given by

$$\Psi = \int_\Omega w\left(\boldsymbol{x}, \boldsymbol{x}'\right) R\left(\boldsymbol{x}', t\right) \mathrm{d}\boldsymbol{x}'. \tag{223}$$

The parameters, τ and α, are temporal scaling parameters. Other parameters present in the model are the degree of heterogeneity in the underlying microscopic network, Δ, the gap junction coupling strength, κ^v, the degree of excitability in the underlying microscopic network, η_0, the synaptic reversal potential, v_{syn}, and the synaptic coupling strength, κ^s. For a full description of the parameters, see Section 6.4.1. Further to the variables already defined, the complex Kuramoto order parameter, Z, can be described by the conformal mapping [124]

$$Z = \frac{1 - W^*}{1 + W^*}, \tag{224}$$

where W^* is the complex conjugate of the value $W = \pi\tau R + iV$. The magnitude of this value, given by $|Z|$, yields the degree of synchrony within the population of neurons. This computed quantity has a range of $[0, 1,]$, where 0 represents complete asynchrony and 1 is full synchrony.

To incorporate space-dependent delays into the model, the non-local interaction term is modified in an analogous fashion to that seen with the previous two models (Sections 7.3.1 and 7.4.1), with the firing rate variable,

R, now becoming dependent on its previous states. This transforms Eq. (223) into

$$\Psi = \int_{\Omega} w\left(\boldsymbol{x}, \boldsymbol{x}'\right) R\left(\boldsymbol{x}', t - s\left(\boldsymbol{x}, \boldsymbol{x}'\right)\right) \mathrm{d}\boldsymbol{x}', \tag{225}$$

where, once again, the delay term takes the form

$$s\left(\boldsymbol{x}, \boldsymbol{x}'\right) = \tau_0 + \frac{\|\boldsymbol{x} - \boldsymbol{x}'\|}{v}. \tag{226}$$

Similarly to the previous models, Eqs. (220) to (222) can be spatially discretised to give

$$\tau \dot{R}_i = 2R_i V_i + \frac{\Delta}{\pi\tau} - R_i\left(g_i + \kappa^v\right), \tag{227}$$

$$\tau \dot{V}_i = \eta_0 - \pi^2\tau^2 R_i^2 + V_i^2 + g_i\left(v_{\text{syn}} - V_i\right), \tag{228}$$

$$\frac{1}{\alpha}\dot{g}_i = -g_i + h_i, \tag{229}$$

$$\frac{1}{\alpha}\dot{h}_i = -h_i + \kappa^s \sum_{j=1}^{N} w_{ij} R_j\left(t - s_{ij}\right)\sigma_j, \tag{230}$$

for $i = 1, \ldots, N$, making note that the second order linear differential operator, Q, has been broken down here to transform Eq. (222) into two first order equations. Likewise, the synchrony for each node can be computed as

$$|Z_i| = \left|\frac{1 - \pi\tau R_i + jV_i}{1 + \pi\tau R_i - jV_i}\right|, \tag{231}$$

for $i = 1, \ldots, N$, where in this case, j represents the imaginary unit.

7.5.2 *Examples*

Here, we explore a selection of example solutions to the delayed next generation neural field model. To start, we fix the delay parameters to have values $v = 10,000\text{mm}\,\text{s}^{-1}$ and $\tau_0 = 0.01\text{s}$ for all simulations (reasoning for this is detailed in Section 7.3.2). It is especially useful to have the constant offset delay, τ_0, present here due to the large number of equations

in the system. The mesh contains 5,304 nodes, thus the total number of equations in the system is 21,216. As locating meaningful solutions via simulation with large-scale delayed equations is very much trial and error due to the inability to accurately perform any kind of numerical stability analyses (at least with the current tools available), solutions that are not of interest can take just as long to compute as solutions that are of interest. The offset delay allows the maximum step size that the solver can reach to be increased, thereby reducing the time it takes to garner a solution.

In comparison to the previous two models explored in this chapter, the hypothesis we wish to explore is whether the next generation model yields more intricate and dynamic patterns that could be conceivably more reflective of real-life observed brain patterns. All dynamic simulations presented here can be viewed in video form at `https://github.com/sammyjp/Thesis-Supplementary-Material`, along with views of other state variables that have not been explicitly included in this section. For simplicity, the figures presented in this section are all from the dorsal perspective; however, other angles are viewable on the supplementary material page.

Figures 47 and 48 depict one period of a dynamic solution to the delay next generation neural field model, with parameters $\tau = 5$, $\eta_0 = 1$, $\alpha = 0.5$, $v_{\text{syn}} = 8$, $\kappa_s = 12$, $\kappa_v = 0.4$ and $\Delta = 0.5$. The first of these figures illustrates the dynamics of the firing rate variable, $R(\boldsymbol{x}, t)$, while the second describes the corresponding synchrony pattern through the computed quantity, $|Z(\boldsymbol{x}, t)|$. Immediately, it is apparent that this solution is formed of localised spots of high firing rate activity that do not remain stationary. To fully appreciate the non-static nature of the solutions, it is advisable to view them at the supplementary material link provided. One period of this specific oscillatory solution lasts approximately 5 seconds. The spots appear to gently rotate in localised regions around the cortex.

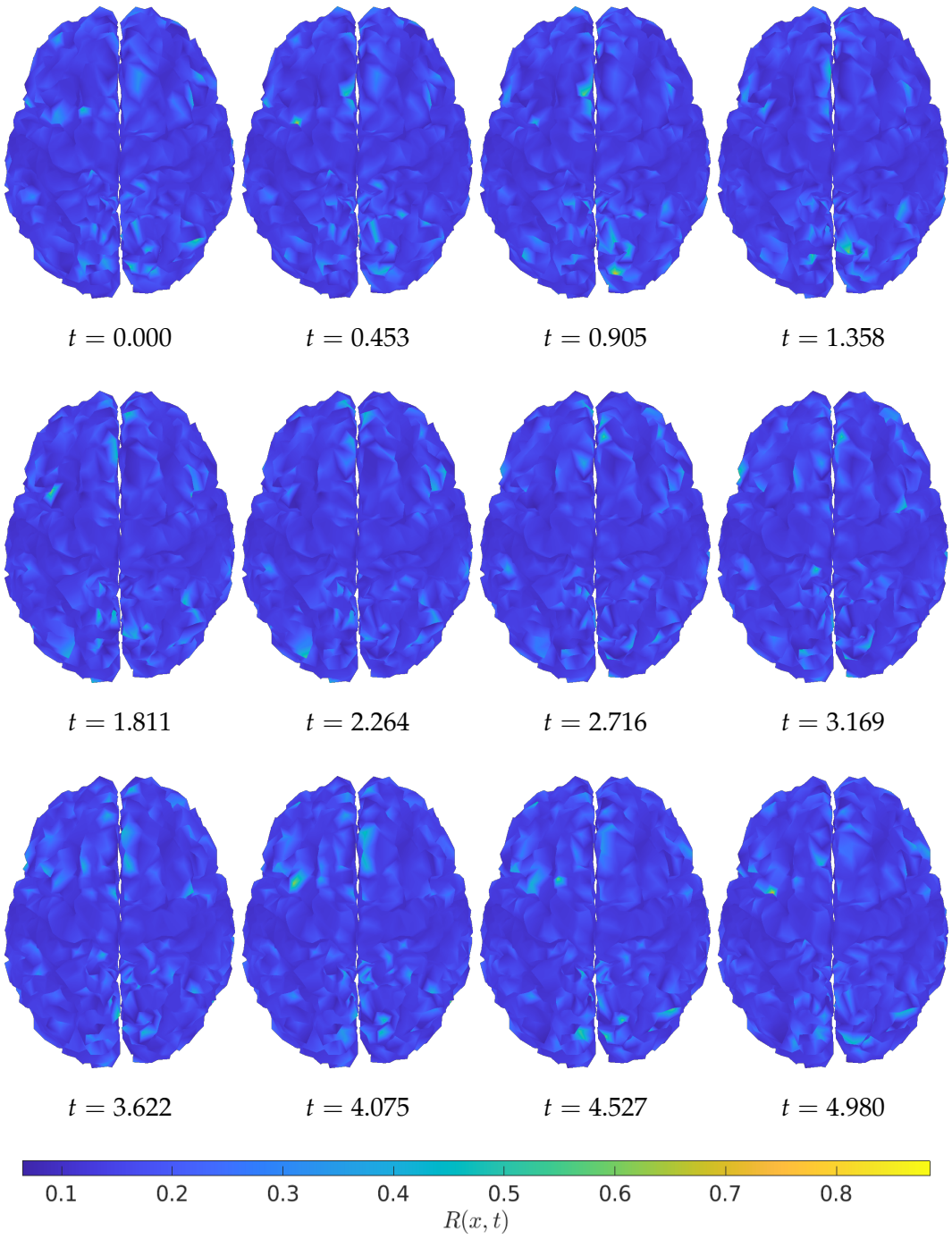


Figure 47: Dorsal view of one period of an oscillatory solution to the next generation neural field model, lasting approximately 5 seconds, arising from a Hopf bifurcation of the steady state. The state variable depicted here is the firing rate variable, $R(\boldsymbol{x}, t)$, and the period is divided into 12 equally spaced frames with times given to three decimal places. Parameter values: $\tau = 5$, $\eta_0 = 1$, $\alpha = 0.5$, $v_{\text{syn}} = 8$, $\kappa_s = 12$, $\kappa_v = 0.4$, $\Delta = 0.5$, $\tau_0 = 0.01$, $v = 10,000$.

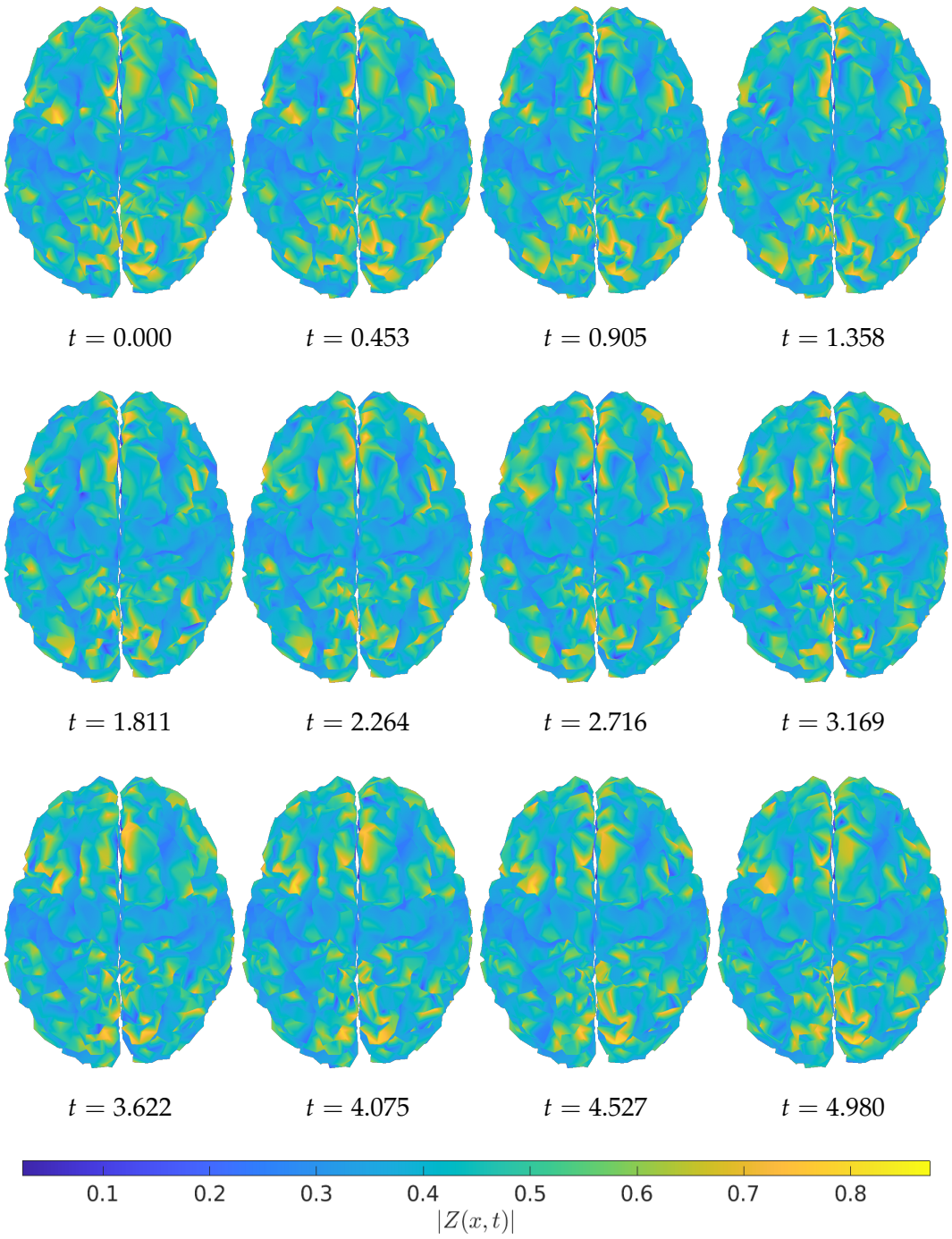


Figure 48: Dorsal view of one period of an oscillatory solution to the next generation neural field model, lasting approximately 5 seconds, arising from a Hopf bifurcation of the steady state. Depicted here is the synchrony variable, $|Z(\boldsymbol{x}, t)|$, and the period is divided into 12 equally spaced frames with times given to three decimal places. Parameter values: $\tau = 5$, $\eta_0 = 1$, $\alpha = 0.5$, $v_{\text{syn}} = 8$, $\kappa_s = 12$, $\kappa_v = 0.4$, $\Delta = 0.5$, $\tau_0 = 0.01$, $v = 10,000$.

A similar observation can also be made for the synchrony, as shown in Figure 48. The majority of the cortex is teal coloured, indicating medium levels of synchrony. However, there are a number of localised spots of high synchrony, as indicated by the yellow regions. When comparing the synchrony to the firing rate, it is extremely evident that the regions where the firing rate is more visibly oscillating are also the areas with the higher degrees of synchrony. Conversely, the horizontal mid section of the cortex (when viewed dorsally) appears to not display much movement in the firing rate solution, which is reflected in the synchrony by it remaining medium-to-low. This example is just one of the intricate solutions that the next generation model gives rise to.

Another is depicted in Figures 49 and 50, this time with a longer oscillatory period of approximately 17.8 seconds. Although it has a longer period than the previous example, this solution has much more distinctive features. There are a larger number of localised spots, and each are clearly visible against the low firing rate that the majority of the cortex exhibits in Figure 49. Dynamically, these spots appear to rotate and move around the cortex, similarly to that of the previous example. The parameters selected for this simulation are $\tau = 10$, $\eta_0 = 2$, $\alpha = 0.8$, $v_{\text{syn}} = 8$, $\kappa_s = 10$, $\kappa_v = 0.8$ and $\Delta = 0.3$. The corresponding synchrony pattern, as visualised in Figure 50, reflects the distinctive features described by the firing rate. There is a much larger degree of synchrony seen across the whole cortex, making note that the minimum synchrony value throughout the entire temporally evolved solution is approximately 0.56. This is reflected well when comparing to the firing rate, as it is clear that the there is localised movement occurring across the entire domain.

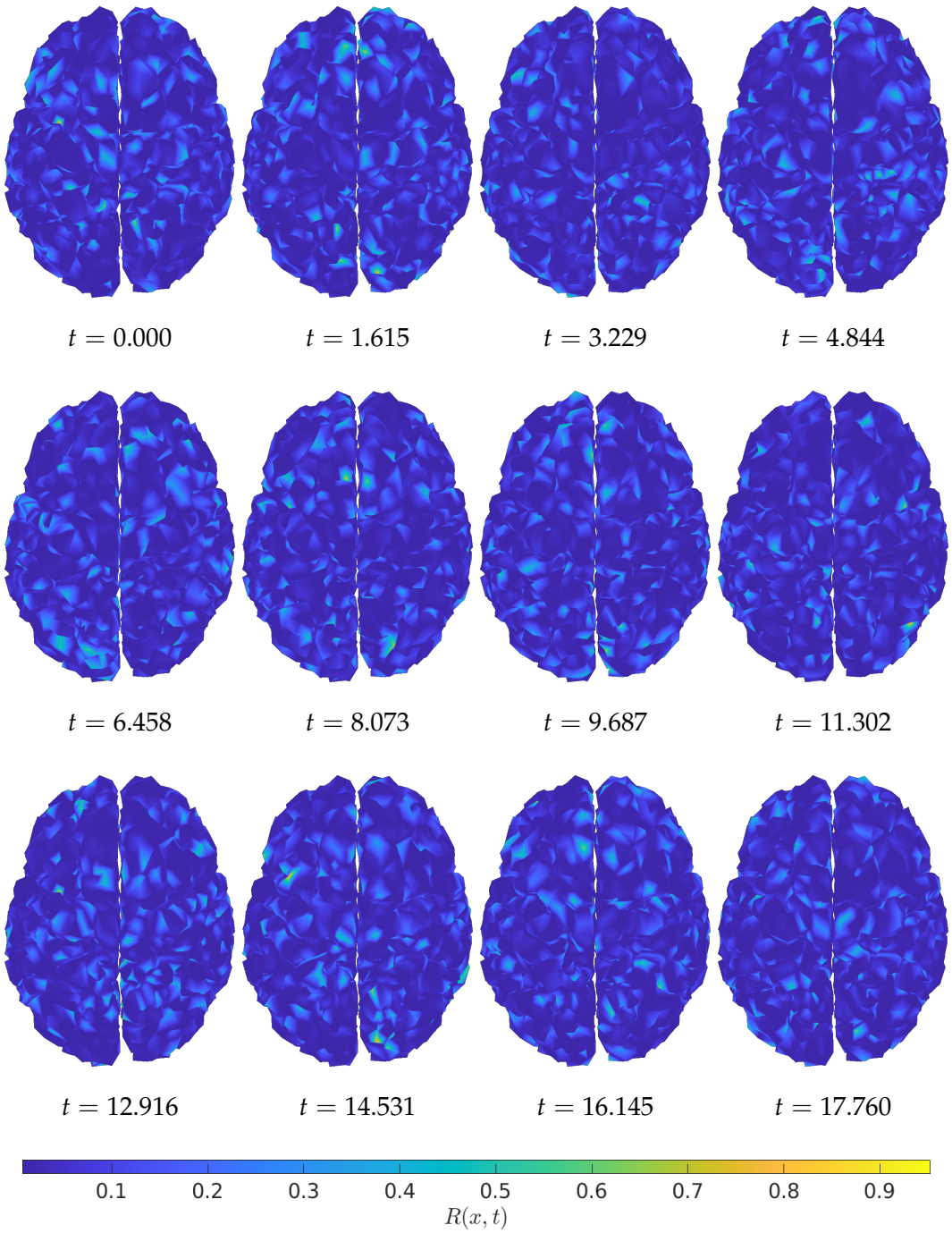


Figure 49: Dorsal view of one period of an oscillatory solution to the next generation neural field model, lasting approximately 17.8 seconds, arising from a Hopf bifurcation of the stationary state. The state variable depicted here is the firing rate variable, $R(\boldsymbol{x}, t)$, and the period is divided into 12 equally spaced frames with times given to three decimal places. Parameter values: $\tau = 10$, $\eta_0 = 2$, $\alpha = 0.8$, $v_{\text{syn}} = 8$, $\kappa_s = 10$, $\kappa_v = 0.8$, $\Delta = 0.3$, $\tau_0 = 0.01$, $v = 10,000$.

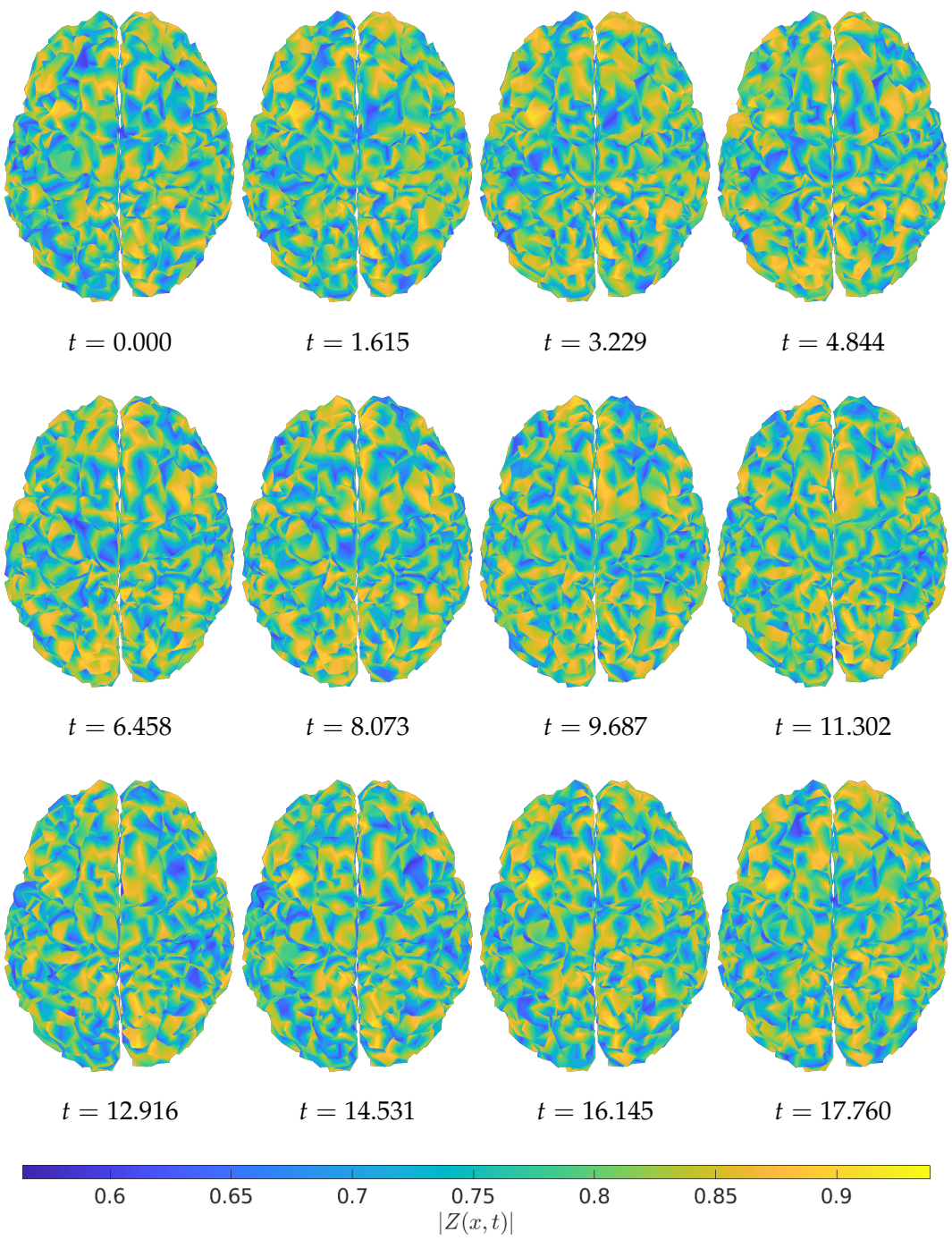


Figure 50: Dorsal view of one period of an oscillatory solution to the next generation neural field model, lasting approximately 17.8 seconds, arising from a Hopf bifurcation of the stationary state. Depicted here is the synchrony variable, $|Z(\boldsymbol{x}, t)|$, and the period is divided into 12 equally spaced frames with times given to three decimal places. Parameter values: $\tau = 10$, $\eta_0 = 2$, $\alpha = 0.8$, $v_{\text{syn}} = 8$, $\kappa_s = 10$, $\kappa_v = 0.8$, $\Delta = 0.3$, $\tau_0 = 0.01$, $v = 10,000$.

The next generation model evidently conforms to our expectation of providing detailed and intricate solutions. It has produced more complex and varied solutions than the standard NFE and NFE with linear adaptation. Other solutions that are not presented here also depict a similar level of intricacy. Although the solutions are alike in their spatial patterning, there are clear variations in the oscillatory period and amplitude. With more detailed tuning of parameters and assessing the significance of realistic physically-based values of each parameter, there is definitely scope to improve on the solutions discovered.

7.6 SUMMARY

This chapter has mostly been a proof of concept to show what is possible when it comes to solving delayed models of neural activity on realistic cortical domains. The three models considered each have their own attributes and challenges when it comes to generating dynamic and detailed solutions.

We first considered the standard NFE, which has been a staple across many mathematical neuroscience studies, and forms the building blocks of a lot of the more advanced models that have been developed since. With the incorporation of real structural brain data into the model, along with path length data for the distribution of delays, we have shown that it is extremely achievable to generate solutions in an efficient and inexpensive manner using the NFESOLVE library. Although due to the time constraints of this work we were unable to present dynamic solutions to the standard delayed NFE, there is definitely scope to explore this model deeper in the future.

Following on from this, we introduced the NFE with linear adaptation. This model is similar to the standard NFE, except that it implements a negative feedback mechanism in the form of a recovery variable. Two example solutions were presented, each depicting a different scenario of

global oscillatory behaviour. Again, similarly to the standard NFE, the model does support different types of oscillatory solutions; however, with this particular data set they proved difficult to locate. It is worth noting that in this section we only made use of a single data set. A future direction this work could be taken in is to explore other data sets from different test subjects and to study how the data affects the solutions yielded from different models. Tuning the delay parameters also has the potential lead to more interesting solutions.

Finally, we explored the next generation NFE pioneered by Coombes and Byrne [36]. The work of Byrne et al. [29] shows the intricate solutions that are possible on a standard rectangular domain, and so it was the aim of this thesis to take their work further by posing the model on the cortical domain. As evidenced by the solutions presented, this model allows for some extremely interesting dynamics on the cortex. Although the example solutions were not directly correlated to any real-life observations, the results yield a great insight into what is possible with this model. Given time to investigate and traverse the capabilities that the next generation model possesses, there is definitely potential to refine the application to realistically recreating observed cortical behaviour.

8

CONCLUSION

To conclude the work discussed in this thesis, we now look back and give an overview of the results discussed in each chapter, before wrapping up by presenting future opportunities and considerations that could take the results even further.

8.1 SUMMARY OF THESIS

The main aim of this thesis was to produce a suite of code that could efficiently solve large-scale delay differential equations, and use this code to explore solutions of delayed neural field models posed on realistic cortical domains.

Chapter 2 introduced and discussed the main background and key concepts required to understand the work explored in this thesis. This began by giving a description of neurons at the cellular level and how they send and receive signals between each other. Following on from this, we examined several models of individual neuron behaviour, with a focus on the groundbreaking work of Hodgkin and Huxley [84], before moving on to discuss models of neural activity at the macroscopic level. This is where we first introduced and broke down the standard NFE, which we have referred to many times across this work, along with the concept of large-scale space-dependent delays. Next, this chapter talked about the methods

behind how macroscopic brain connections are measured in practice. Here, we presented the differences between structural and functional MRI, and how organisations such as the Human Connectome Project [172, 173] have gathered and published a vast amount of brain data, some of which we make use of in our simulations. Finally, we introduced the fundamentals of numerical quadrature rules for approximating integrals. Here, we met Gauss-Legendre quadrature, which is a key numerical rule that we employ in Section 3.3.

The following chapter illustrated a selection of analytical and numerical techniques that are an important prerequisite to the work that follows in the latter chapters. The first area considered was the construction and analysis of an arbitrary non-linear network. Here, we presented the linear stability analysis for a network whose dynamics are governed by generic non-linear functions, in order to easily compute eigenvalues for any given network model of neural activity. This was especially useful as it allowed for information to be yielded quickly on the behaviour of a system for a given parameter regime. Also included here was the analysis for the incorporation of delays into the network. We made use of this in Chapter 6 where we considered solutions to a variety of different network models, with the intention of locating Hopf regimes that yield oscillatory dynamics. Moving on from this, we discussed the spatial discretisation of continuum models of neural activity so that they may be evolved temporally via numerical methods. Here, we covered the definition of the spatial mesh, along with quadrature rules for numerically approximating the integral terms present in neural field models. We then illustrated how these techniques can be applied to the NFE, via the Nyström method, to discretise it into a system of ODEs. To verify that the disretisation scheme yields solutions that match theoretical expectations with respect to error, a convergence analysis was undertaken. This involved using specially constructed problems for which an analytical solution is known

so that the numerical solution could be compared to the true solution. As demonstrated, the chosen discretisation methods perform as expected.

Chapter 4 gave an in-depth look at the design of our highly efficient suite of differential equation solvers: NFESOLVE. This began by laying out the drawbacks of existing solvers and how the NFESOLVE could fill the gaps that these solvers have. We then moved on to discuss the choices made prior to beginning development about the tools and methodologies we would use to build the library. The remaining sections of this chapter outlined the design of each module that makes up the library, starting with the classes for handling everything to do with storing and numerically integrating over a geometric mesh. This was followed by a description of the design of the ODE solver suite, before moving on to exploring the nuances of designing an efficient DDE solver package. As part of this, we introduced our sparse DDE solver module, which was designed to eliminate unnecessary computations from the solution process. This was one of the major areas where we were able to streamline and improve upon existing DDE solvers. Finally, we presented how the code may be sped up by making use of OpenMP parallelisation [43] when working with large systems of equations containing multiple delays.

After giving this explanation of the design of the NFESOLVE library, Chapter 5 then moved on to exploring several different validation examples with known results in order to prove that our code performed to the expected standard. Firstly, an error analysis was undertaken to demonstrate that each ODE and DDE solver yield numerical solutions which converge to their corresponding analytical solutions at the rates depicted in the names of each solver (i.e., proving that *RungeKutta3Solver* is a third order solver). To follow this, the next section looked at solutions to the standard NFE on a spherical domain. This work was based on results by Visser et al. [177] and was selected here as a validation example due to the analytical

results available, along with the spherical geometry being treated as a very naive cortex, paving the way for the work explored in Chapter 7. We then considered the one-dimensional NFE with the presence of a single constant delay, before finally moving on to discussing various neural field examples with full-scale space-dependent delays. Here, we also talked about the most efficient way to set up the sparse solvers in the NFESOLVE library to solve neural field models. This chapter allowed us to demonstrate that the NFESOLVE library performs to the specifications dictated, and that it can applied to solve large-scale models of neural activity with multiple delays.

Chapter 6 proceeded to explore the relationship between structural and functional connectivity in networks of neural masses. In this chapter, we first presented a set of brain data supplied by the HCP. This included structural connectivity, functional connectivity, and path length data. We then explained the theory behind using structure to predict function via eigenmode fitting, based on work by Tewarie et al. [164]. To illustrate this practically, we considered two different models of neural activity, starting with the Wilson-Cowan model. Using techniques introduced in Chapter 3, we formulated eigenvalue equations for the model in both the absence and presence of delays in order to inform on the eigenmodes that were unstable for a given parameter regime. This allowed us to locate parameter regimes which yielded solutions whose dynamics were governed by specific eigenmodes which the fitting analysis dictated would be most reflective of the functional connectivity. We then introduced a next-generation neural mass model, pioneered by Coombes and Byrne [36], and repeated the analysis process to again locate parameter regimes which would yield relevant solutions. The work undertaken here showed how the next-generation model can produce much more desirable solutions than models such as the Wilson-Cowan model, while simultaneously being much more grounded in biological reality.

The final chapter of this thesis took a deep dive into the solution of delayed neural field models posed on realistic cortical domains. It began by introducing the data, supplied by the HCP, which was incorporated into the various models considered. This included the cortical mesh, structural connectivity, and path length data. The first model presented was the standard delayed NFE. Here, we illustrated some of the steady state solution patterns which the model naturally supports. Following this, we moved on to a more advanced model of neural activity: the NFE with linear adaptation. The negative feedback mechanism in this model allows it to naturally support dynamic solutions which arise via a Hopf bifurcation. In this section, we build upon the steady states found for the standard NFE in order to generate several different globally oscillatory solutions. Finally, we re-introduced the next-generation model described in Chapter 6, and demonstrated how it can be adapted from a neural mass model to a neural field model. As predicted, this model yielded considerably more interesting dynamic solutions with highly intricate patterning. This leaves huge scope for the future of dynamic brain modelling, as the tools to efficiently solve realistic large-scale delayed models of neural activity now sit at the fingertips of scientific researchers.

8.2 FUTURE WORK

The work undertaken in this thesis has only just scratched the surface of what is now possible with modern algorithms, technology, and models of neural activity. There are a number of directions in which this work could be taken in the future, with areas from Chapter 4, Chapter 6, and Chapter 7, all having individual scope for future considerations. This section outlines some of these areas where there is a strong potential to build upon the current work.

8.2.1 *NFESOLVE Improvements*

The NFESOLVE library is still in its infancy when it comes to the features it could possess and the improvements that could be made to the existing code. Here, we will outline some of these improvements and how they would benefit the library.

Quadrature Library

As discussed in Section 4.3.2, the current implementation of the *Quadrature-Library* namespace only supports vertex quadrature rules. A key improvement that could be made is to add higher order quadrature rules, such as Gaussian quadrature. This would not be a difficult task to approach, and the only reason that it is not presently a part of the library is due to vertex quadrature rules being sufficient for the work presented in this thesis. Gaussian quadrature, as traditionally implemented for interval domains, can be analogously extended to domains made up of triangular or quadrilateral elements [49].

Automatic Delay Matrix Population For Sparse Solvers

At present, the sparse delay solvers that are part of the NFESOLVE library (see Section 4.6) require a sparsity pattern to be passed in to the constructor, so that the solver knows which entries in the delay state matrix require computing. A future consideration for the library would be to remove the need for the user to pre-compute this sparsity pattern and instead have the solver automatically generate the sparsity pattern on the first iteration of the time-stepping process. One way of doing this could be by designing a custom data structure that records the accessed indices on the first pass until it has built an entire map of the elements that are non-zero. Once the sparsity pattern has been built then it would be used to instantiate a sparse matrix that follows the current implementation design. Although

this change will have no direct impact to the time-stepping algorithm, it is definitely a design choice that would improve user experience and make it easier for someone who did not have knowledge of sparse matrices, or their use in delay differential equation solvers, to use the NFESOLVE library.

Linear Algebra Library

As discussed in Section 4.2, the NFESOLVE library is built upon the data structures provided by the Armadillo [145, 146, 147] linear algebra library. This is a fantastic library that offers a variety of vector and matrix structures, as well as several linear algebra operations, while providing a MATLAB-like syntax. However, due to the way it is designed, there are slight negative impacts to performance when compared to some of the other linear algebra libraries that are openly available. Armadillo was originally selected due to its similarities to MATLAB (which is commonly used by researchers in the area of applied mathematics), and its wide use across a number of other tools and libraries. At the time of development, it met all the necessary requirements to build the NFESOLVE library and was extremely easy to integrate with, making it a good starting point for developing a brand new suite of solvers. However, upon further research into the other tools available that provide the same/similar functionality, there are considerations that could be made for future implementations of the NFESOLVE library in order to improve performance.

One such library that is frequently used across a variety of applications is Eigen [76]. It boasts extremely fast and efficient computations, and possesses all the features that would be required in the NFESOLVE library. It does not rely on a back-end implementation of BLAS and LAPACK like Armadillo does, instead containing its own built-in optimised operations and subroutines. Ultimately, the key selling point of the NFESOLVE library is its ability to solve large-scale delay differential equations in an efficient time frame. Although Armadillo offers a MATLAB-like syntax, which is useful

for programmers who are new to C++, the main focus of the NFESOLVE library should be to achieve the best performance that it can offer.

Stochastic Differential Equations

Although not mentioned as part of the scope of this thesis, an extremely rich area of opportunity to expand the work presented here is the world of stochastic neural modelling. Naturally, the signals that are measured by MRI and EEG/MEG contain some degree of noise. The models we have considered thus far are purely deterministic and do not account for any sort of noise that would likely be present in the signals that we are attempting to recreate. Stochastic modelling of biological phenomena has been a long-studied branch of mathematics, with authors such as Bressloff [21, 22] providing a historical insight into the incorporation of stochastic processes into models of neural activity.

When it comes to the numerical solution of stochastic differential equations (SDEs), there exist several numerical methods that can be implemented to facilitate this. As an initial starting point, the Euler-Maruyama scheme [121, 107] is a commonly used method for time-stepping SDEs. The NFESOLVE library is designed in such a way that it is simple to expand and introduce new solvers and features. A key area of improvement for the future would be to create a suite of SDE solvers to complement the existing ODE and DDE solvers. Note that comparatively little work has been done with regards to developing numerical schemes for approximating the solutions of stochastic delay differential equations (SDDEs). This is very much an area of interest for future work, as when combined with the sparse delay solvers available in the NFESOLVE library, methods for solving SDDEs could see a significant increase in efficiency and open up a world of possibilities for the neural modelling communities.

8.2.2 *Eigenmode Fitting for Neural Mass Networks*

Chapter 6 demonstrated techniques for predicting functional connectivity using the eigenmodes of structural connectivity matrices. As with all of the examples considered in this thesis, one of the biggest areas which could be improved upon in future is the amount of different data that could be utilised in the simulations. The results presented in this chapter only considered the data from a single HCP subject; however, there is astronomically more data available, from multiple subjects, that could be explored to give a wider array of results. Similarly, we only consider two models of neural activity in this chapter. There are a number of variations and other models that exist which may present solutions more reflective of the observed functional connectivity which we are attempting to recreate. The eigenmode fitting process is independent of the model which the data is being incorporated into, thereby not guaranteeing that the models considered will actually yield the desired solutions.

Another point for consideration is the method by which the eigenmodes of the structural connectivity matrices are constructed. In this chapter, we use the cosine of the phase differences between each eigenvector, which is preferred over the outer product calculation used by Tewarie et al. [164]. However, there may be other constructions which yield better simulation results. Also related to this are the pre-processing techniques that the structural matrices undergo before being incorporated into the models. As detailed in Section 6.3.3, in order to easily compute the stability of the system a row-sum normalisation condition must be applied. Although this makes analysis much simpler, especially in the presence of delays, one of the main drawbacks of using a row-sum normalised matrix is that it heterogeneously transforms the data away from its original symmetric form. This can drastically affect the solutions that the models yield and may make the prediction of functional connectivity worse than if the original matrix was used prior to any row-sum normalisations. With the ability to be able to accurately

predict stability of a model using the non-normalised matrix, a future enhancement could be to explore the goodness of fit of the models with this original structural connectivity.

Computing the BOLD Signal

Currently, we have only considered the functional connectivity of the raw activity signals generated by the models. In practice, functional connectivity generated by an fMRI is measured via the BOLD signal [74, 169] (see Section 2.4.2 for more details). This differs from the activity signals seen in the numerical solutions presented. In order to yield a representation of the BOLD signal from these activity signals, the solution can be passed through a haemodynamic filter known as the Balloon-Windkessel model [69]. This is given by the set of equations

$$\frac{\mathrm{d}}{\mathrm{d}t}x(t) = \varepsilon u(t) - kx(t) - \gamma\left(f(t) - 1\right), \tag{232}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}f(t) = x(t), \tag{233}$$

$$\tau\frac{\mathrm{d}}{\mathrm{d}t}v(t) = f(t) - v^{\frac{1}{\alpha}}(t), \tag{234}$$

$$\tau\frac{\mathrm{d}}{\mathrm{d}t}q(t) = \frac{f(t)}{\rho}\left(1 - (1-\rho)^{\frac{1}{f(t)}}\right) - q(t)\,v^{\frac{1}{\alpha}-1}(t), \tag{235}$$

where u is a neural activity signal, x is the vasodilatory signal, f is blood inflow, v is blood volume, and q is deoxyhaemoglobin content. The parameters ε, k, γ, τ, α, and ρ, represent the activity signal coupling strength, rate of signal decay, rate of flow-dependent elimination, haemodynamic transit time, Grubb's exponent, and resting oxygen extraction fraction, respectively. Typical values and explanations of each of these parameters can be found in [48]. Given the blood volume and deoxyhaemoglobin content state variables from the above model, the BOLD signal, $y(t)$, may then be computed via

$$y(t) = V_0\left[k_1\left(1 - q(t)\right) + k_2\left(1 - \frac{q(t)}{v(t)}\right) + k_3\left(1 - v(t)\right)\right]. \tag{236}$$

Here, V_0 is the resting blood volume fraction and the parameters k_1, k_2, and k_3 are dimensionless constants which reflect baseline physiologial proper-

ties of brain tissue [80]. Utilising the BOLD signal to then generate a functional connectivity matrix may yield results that differ from those presented in Chapter 6. As a piece of future work, this definitely holds value and may allow an even greater understanding of how structure can be used to predict function.

8.2.3 *Models Posed on Cortical Domains*

The work presented in Chapter 7 provides a strong foundation in demonstrating how models of neural activity can be applied in a realistic geometric setting with the incorporation of real patient data. The NFE-SOLVE library made it simple to generate solutions to the models in the presence of space-dependent axonal delays. However, the work undertaken as part of this thesis just scratches the surface of what is possible, and there is clearly a plethora of opportunities that this work has opened up for the future.

One of the main limitations discussed in Chapter 7 is the inability to accurately perform stability analyses on the delayed versions of the chosen models. Due to the large number of mesh points required to preserve cortical detail in the mesh, along with the transcendental nature of the stability equations for delayed systems, locating parameter regimes under which a desired bifurcation has occurred is near impossible via traditional methods. The main tool at our disposal for locating different solution types, at least for the results shown, was trial and error. The example solutions to the delayed standard neural field model, presented in Section 7.3.2, were both steady state solutions. Ideally, a future direction this work could be taken in is to locate other, more interesting solutions that arise from this model, with a focus on dynamic patterns. Although the current data assumptions may mean that there does not exist dynamic solutions in this context, there is still scope to explore and categorise the

range of solutions this model potentially has to offer.

Similarly, for both the other models explored, this work only presented two examples of dynamic solutions for each in order to show a glimpse into the possibilities that arise when they are posed on realistic cortical domains. These solutions were interesting, especially for the next generation neural field model; however there is a considerable amount of work that can be done in order to attempt to yield solutions more reflective of observed brain patterns. Building upon the work achieved by Byrne et al. [29], who study the model in both one and two dimensions, there is scope to extend this to a cortical domain to record the effects of each parameter on the patterns generated, and attempt to replicate EEG/MEG signals.

Another hugely important consideration for future development of the work in this thesis is the exploration of other data sets and their effects on the solutions of different models. All of the data used in Chapter 7 came from a single subject; however, the HCP have a large amount of data from many different individuals, all of which could yield different and more interesting results when incorporated into models of neural activity. There is also opportunity to look at different types of data, such as fMRI, along with attempting to recreate different neurological phenomena, such as beta rebound currents [27].

A

TECHNICAL BACKGROUND

A.1 RUNGE-KUTTA SCHEMES FOR ORDINARY DIFFERENTIAL EQUATIONS

The numerical evaluation of differential equations has been a long studied area of mathematics, dating back to the 18th century. First presented in 1768 [60], this started with Euler's method for solving initial value problems (IVPs) of the form

$$y'(t) = f(t, y(t)), \quad y(t_0) = y_0, \tag{237}$$

where $t \in \mathbb{R}$, $y \in \mathbb{R}^n$, $f : \Omega \subset \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$, the domain Ω is an open set of $\mathbb{R} \times \mathbb{R}^n$, and the point (t_0, y_0) lies in Ω. Here, in Appendix A.1, we shall outline formulae developed to solve differential equations and the accompanying adjustments that can be incorporated to allow for continuous output and step size adaptation. Appendix A.2 will then build on this work to present methods for numerically solving differential equations with the inclusion of delays.

A.1.1 *Explicit Runge-Kutta Formulae*

For a fixed step $h \in \mathbb{R}$, Euler's method is given by

$$y_{n+1} = y_n + hf(t_n, y_n), \tag{238}$$

where $t_n = t_0 + nh$, for $n = 0, 1, 2, \ldots$, and y_n is the approximate solution to the IVP at point t_n. This gives a first order approximation of the solution [7], i.e.

$$\max_n |y(t_n) - y_n| \leq Ch, \tag{239}$$

for some constant $C \geq 0$. Using big O notation, we say that the scheme is $\mathcal{O}(h)$.

In 1895, and again in 1900, Runge [144] and Heun [82] furthered Euler's work by adding in additional steps to the method. Kutta [109] then formulated, in 1901, what is known today as the Runge-Kutta method. In k notation, a general s-stage ($s \in \mathbb{Z}^+$) explicit Runge-Kutta (ERK) scheme is written as

$$\begin{aligned}
k_1 &= f(t_n, y_n), \\
k_2 &= f(t_n + c_2 h, y_n + h a_{21} k_1), \\
k_3 &= f(t_n + c_3 h, y_n + h a_{31} k_1 + h a_{32} k_2), \\
&\vdots \\
k_s &= f\left(t_n + c_s h, y_n + h \sum_{j=1}^{s-1} a_{sj} k_j\right), \\
y_{n+1} &= y_n + h \sum_{i=1}^{s} b_i k_i,
\end{aligned} \tag{240}$$

where the coefficients a_{ij}, b_i, c_i, for $i = 1, 2, \ldots, s$ and $j = 1, 2, \ldots, i-1$, are real numbers and $c_1 = 0$. The coefficients c_i typically satisfy the conditions $c_i = \sum_{j=1}^{i-1} a_{ij}$, so that all the points where f is evaluated are first order approximations to the solution [77]. A Runge-Kutta method is said to be of order p (or $\mathcal{O}(h^p)$), if for sufficiently smooth problems,

$$\|y(t_{n+1}) - y_{n+1}\| \leq Kh^{p+1}, \tag{241}$$

for some constant $K \geq 0$. The classical Runge-Kutta method [109] is a 4th order method comprised of four stages and is given as

$$\begin{aligned} k_1 &= f\left(t_n, y_n\right), \\ k_2 &= f\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_1\right), \\ k_3 &= f\left(tn + \frac{1}{2}h, y_n + \frac{1}{2}hk_2\right), \\ k_4 &= f\left(t_n + h, y_n + hk_3\right), \\ y_{n+1} &= y_n + \frac{1}{6}h\left(k_1 + 2k_2 + 2k_3 + k_4\right). \end{aligned} \tag{242}$$

In 1946, from work by Butcher [24], it became convention to write the coefficients of a Runge-Kutta scheme in a tableau, now called a Butcher tableau.

$$\begin{array}{c|cccc} c_1 & & & & \\ c_2 & a_{21} & & & \\ c_3 & a_{31} & a_{32} & & \\ \vdots & \vdots & \vdots & \ddots & \\ c_s & a_{s1} & a_{s2} & \dots & a_{s,s-1} \\ \hline & b_1 & b_2 & \dots & b_s \end{array} . \tag{243}$$

The Runge-Kutta method can also be written using Y notation as

$$Y_i = y_n + h\sum_{j=1}^{i-1} a_{ij} f\left(t_n + c_i h, Y_j\right), \qquad i = 1, \dots, s, \tag{244}$$

$$y_{n+1} = y_n + h\sum_{i=1}^{s} b_i F\left(t_n + c_i h, Y_i\right). \tag{245}$$

There exist many different Runge-Kutta schemes of varying orders and number of stages [62, 139, 77, 94].

A.1.2 *Continuous Output*

Pioneered by Horn in 1983 [88], *continuous output* Runge-Kutta schemes allow solution values to be calculated at points that do not fall on the discretisation grid, i.e., for an interval $[t_n, t_n + h]$, we can compute $t_n + \theta h$ for

some $0 \leq \theta \leq 1$. This can be done using polynomials in θ as the b coefficients in the Runge-Kutta scheme [152]. Starting from a standard s stage Runge-Kutta scheme with coefficients a_{ij}, b_i and c_i, if additional stages are added such that the method now has s^* stages, then the following formula can be constructed:

$$k_i = f\left(t_n + c_i h, y_n + h\sum_{j=1}^{i-1} a_{ij}k_j\right), \qquad i = 1, \ldots, s^*, \tag{246}$$

$$u_n(\theta) = y_n + h\sum_{i=1}^{s^*} b_i(\theta)\, k_i, \tag{247}$$

where $b_i(\theta)$ are polynomials in θ to be determined such that

$$u_n(\theta) - y(t_n + \theta h) = \mathcal{O}\left(h^{p^*+1}\right). \tag{248}$$

The construction of these polynomials is explained in Hairer, Norsett and Wanner [77] and Owren and Zennaro [132]. However, we shall not pursue this here. A more efficient method is to use a two point Hermite interpolant [151, 163], as this does not require the use of the k stages of the Runge-Kutta scheme and can therefore be called at any time. This formula is given as

$$y(t_n + \theta h)) = (1-\theta)\, y_n + \theta y_{n+1} + \theta(\theta-1)\left((1-2\theta)(y_{n+1} - y_n) + (\theta-1)\, h y'_n + \theta h y'_{n+1}\right), \tag{249}$$

where y'_n represents the approximation of the derivative of the solution (with respect to t) at time t_n, i.e., $y'n \approx y'(t_n)$. This may be computed at each Runge-Kutta stage as $y'_n = f(t_n, y_n)$. It can be shown that Eq. (249) is a special case of Eq. (247) [77]. For any underlying scheme of order $p \geq 3$, this will give a 3rd order continuous output Runge-Kutta scheme [77]. This method also guarantees that the continuous output solution will be globally $\mathcal{C}^1$ continuous, whereas with the polynomials this is not always the case.

A.1.3 *Embedded Runge-Kutta schemes*

Suppose that instead of a fixed step size h, we wish to utilise a method that can adapt the step size in order to compute solutions more efficiently with respect to their behaviour. In particular, we aim to select, at each step, the smallest step size that meets a user-specified error tolerance. This can be done by computing a second approximation $\hat{y}_n$ along with the approximation y_n and then using these to compute an estimate of the *local error* at that point. Let us consider these approximations such that y_n is of order p and $\hat{y}_n$ is of order $\hat{p}$ and they are both computed using the same function evaluations. This is referred to as a $p\,(\hat{p})$ *embedded* Runge-Kutta method. The method is written out as

$$k_i = f\left(t_n + c_i h, y_n + h\sum_{j=1}^{i-1} a_{ij}k_j\right), \qquad i = 1, \ldots, s, \tag{250}$$

$$y_{n+1} = y_n + h\sum_{i=1}^{s} b_i k_i, \tag{251}$$

$$\hat{y}_{n+1} = y_n + h\sum_{i=1}^{s} \hat{b}_i k_i. \tag{252}$$

An extra line is added to the Butcher tableau for the $\hat{b}$ coefficients.

$$\begin{array}{c|cccc} c_1 & & & & \\ c_2 & a_{21} & & & \\ c_3 & a_{31} & a_{32} & & \\ \vdots & \vdots & \vdots & \ddots & \\ c_s & a_{s1} & a_{s2} & \ldots & a_{s,s-1} \\ \hline & b_1 & b_2 & \ldots & b_s \\ & \hat{b}_1 & \hat{b}_2 & \ldots & \hat{b}_s \end{array} \tag{253}$$

The first methods of this type were developed by Merson [123], Ceschino [30] and Zonneveld [185] in the late 1950s and early 1960s. In the case where

$\hat{p} > p$, the lower order approximation is propagated forward as the solution and the higher order approximation is used to estimate the local error by the quantity $y_{n+1} - \hat{y}_{n+1}$. However, when $\hat{p} < p$, it is the higher order approximation that is used as the solution value and the lower order approximation is used purely for step size adaptation, abandoning the concept of 'error estimation' [77, 150, 100]. This is referred to as *local extrapolation* [149, 52] and tends to make the method very accurate with regards to global error [9]. Typically, the approximation $\hat{y}_n$ has either order $\hat{p} = p + 1$ or $\hat{p} = p - 1$. As an example, for any 3rd order method [77] with $s = 3$ stages, the coefficients $\hat{b}_i,\ i = 1, \ldots, 3$, required to give a 2nd order method can be given by

$$\hat{b}_1 = 1 - \frac{1}{2c_2}, \qquad \hat{b}_2 = \frac{1}{2c_2}, \qquad \hat{b}_3 = 0. \tag{254}$$

A.1.4 *Step size adaptation*

For the step size adaptation, we want the quantity $y_{n+1} - \hat{y}_{n+1}$ to satisfy, componentwise,

$$\left| y^i_{n+1} - \hat{y}^i_{n+1} \right| \leq sc_i, \tag{255}$$

where

$$sc_i = \mathrm{ATol}_i + \max\left(\left| y^i_n \right|, \left| y^i_{n+1} \right| \right) \cdot \mathrm{RTol}_i, \quad i = 1, \ldots, m, \tag{256}$$

and m is the number of equations in the system. Here *ATol* and *RTol* represent absolute and relative tolerances respectively. A measure of the error is taken as

$$err = \sqrt{\frac{1}{m} \sum_{i=1}^{m} \left(\frac{y^i_{n+1} - \hat{y}^i_{n+1}}{sc_i} \right)^2}, \tag{257}$$

which is used to compute a new step size and determine whether it is accepted. Ceschino [31] tells us that optimal step size is

$$h_{\mathrm{opt}} = h \cdot \sqrt[p]{\frac{1}{err}}. \tag{258}$$

However, to increase the probability that the error the next time will be acceptable, we multiply by a safety factor, *fac*. This is usually taken to

be between 0.8-0.9, as taking the optimal step size on its own will likely yield errors that are slightly too large for the desired tolerance [155]. It is important that h is prevented from increasing or decreasing too quickly, hence a minimum and maximum factor that h can be multiplied by is also specified. These are denoted in the formula below by *facmin* and *facmax*, respectively. The new step step size is computed using

$$h_{\text{new}} = h \cdot \min\left(facmax, \max\left(facmin, fac \cdot \sqrt[p]{\frac{1}{err}}\right)\right). \tag{259}$$

If $err \leq 1$, then the new step size is accepted and used in the next step of the Runge-Kutta scheme. If $err > 1$, then the new step size is rejected and Eq. (259) is repeated using the newly computed h_{new} in the place of h. In the event of a rejected step size it is also advisable to set $facmax = 1$ for the new computation. This is because if a step has been rejected then it is likely that the solution changes behaviour and so caution must be taken not to increase the step size too much [155].

Amongst the most commonly used embedded Runge-Kutta methods are the Dormand-Prince 5(4) scheme [52] used in MATLAB's ode45 solver, the Bogacki-Shampine 3(2) scheme [20] used in MATLAB's ode23 solver, and others outlined by Fehlberg in [62].

A.2 RUNGE-KUTTA SCHEMES FOR DELAY DIFFERENTIAL EQUATIONS

In this section, we give an introduction to delay differential equations (DDEs), also referred to in older literature as differential-difference equations. We will then present some of the methods that exist to solve these types of equations, building on work seen in the previous section. Finally, we show how these methods can be utilised to build a powerful DDE solver; and subsequently, perform convergence tests to show that the observed numerical error is congruent with the theoretical error approximation.

A.2.1 *Delay Differential Equations*

Modelling using DDEs has become a common tool for many applications including, but not limited to, biological phenomena such as population dynamics, immunology, physiology, neural networks and epidemiology [141, 157]. A famous example is the delayed logistic equation for population growth, developed by Hutchinson in 1948 [89]. The standard logistic equation is given by

$$N'(t) = N(t)\left[b - aN(t)\right], \tag{260}$$

where $a, b > 0$. Hutchinson noticed that population densities influence birth rates at later times due to developmental and maturation delays. This led him to introduce a delay into the equation, giving

$$N'(t) = N(t)\left[b - aN(t-r)\right], \tag{261}$$

where $r > 0$.

In [17], Bellman and Cooke illustrate the general theory behind DDEs. A *retarded* DDE has the form

$$y'(t) = f\left(t, y(t), y(t-\tau_1), y(t-\tau_2), \ldots, y\left(t-\tau_q\right)\right), \qquad t \geq t_0, \tag{262}$$

$$y(t) = \varphi(t), \qquad t \leq t_0. \tag{263}$$

The delays $\tau_1, \ldots, \tau_q$ can be constant ($\tau = c \in \mathbb{R}$), time dependent ($\tau = \tau(t)$), or state dependent ($\tau = \tau(y(t))$). For the purposes of this work we shall only consider constant delays; however, methods for handling time dependent and state dependent delays can be found in Bellen and Zennaro [16]. The function $\varphi : \mathbb{R} \to \mathbb{R}^n$ is known as the *history function* as it provides a solution for when t is less than the initial time t_0.

There also exists a type of DDE called a *neutral* DDE. These can be dependent on the past states of the derivative y' as well as the past states of the solution y. Again, we do not consider those in this work, but they are discussed in Hale [78].

DDEs can be solved numerically in a similar way to ODEs, except the delay states also need to be evaluated for use in the computation of the solution. This usually requires some form of interpolation as the delay terms are unlikely to fall on the stages of the standard Runge-Kutta timestepper.

A.2.2 *Natural Runge-Kutta Method for DDEs*

If the delay terms do fall on the Runge-Kutta stages then we can employ a method known as the Natural Runge-Kutta (NRK) method for delay differential equations [16]. Suppose that for a non-adaptive method with fixed step size h, we have a single constant delay τ such that $\tau = \kappa h$ for some integer κ. Having such a condition means that the delay terms will always fall on the stages of the Runge-Kutta scheme, hence no interpolation is required. In Y notation, the scheme is given by

$$Y^i_{n+1} = y_n + h \sum_{j=1}^{i-1} a_{ij} f\left(t_n + c_i h, Y^j_{n+1}, Y^j_{n+1-\kappa}\right), \quad i = 1, \dots, s, \tag{264}$$

$$y_{n+1} = y_n + h \sum_{i=1}^{s} b_i f\left(t_n + c_i h, Y^i_{n+1}, Y^i_{n+1-\kappa}\right). \tag{265}$$

When n is less than κ, the delay terms fall in the history function $\varphi(t)$. This is extended analogously for multiple delays; however, all the delays must be integer multiples of h. The downside to this method is that it requires all the Runge-Kutta stages for each step (or at least as far back as the $(n+1-\kappa)$th step) to be saved in memory. Otherwise, they need to be recomputed again at the moment they are needed. It is also very restrictive in terms of the time step that can be chosen, and does not allow for step size adaptation. The method becomes highly impractical when working with large systems of equations and/or a large number of delays [77].

A.2.3 *Continuous Output Methods for DDEs*

The NRK method is good for small systems with a single delay and fixed step size, but what if a more flexible method is desired? As illustrated in Figure 51, suppose that a delay state (red dot) falls arbitrarily between two previously computed solution points (blue dots).

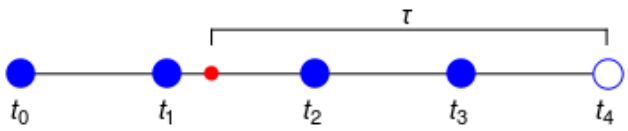


Figure 51: Example delay state timeline where the closed blue dots represent computed solution points, the open blue circle is the current solution point being computed, and the red dot is an arbitrary example of where a delay state may fall between two previously computed solution points.

The progression of the scheme is dependent on computing this delay state. As the delay state is bounded by known solution points, an interpolant can be applied [93, 54]. The order of the interpolant, and hence the number of support points required, would have to be varied in accordance with the order of the Runge-Kutta scheme to achieve the desired order of the overall method [128, 95]. Given a pth order underlying Runge-Kutta scheme with a qth order interpolant for the delay terms, the overall order of the DDE scheme is $\min(p, q)$ [133]. Continuous output schemes, as presented in Appendix A.1.2, can be utilised to provide an approximation to the delay state that maintains the order of the scheme without requiring many extra computations. This can be seen in Shampine and Thompson [154], where they discuss the methods behind MATLAB's dde23 solver. These methods are also more flexible as they allow for the underlying Runge-Kutta scheme to be adaptive.

A.3 SPARSE MATRIX STRUCTURES

There are several storage methods for storing sparse matrices, such as *coordinate* (COO) format, *compressed sparse row* (CSR) format and *compressed sparse column* (CSC) format [136, 106]. Although the most commonly used format in general is CSR, this project mainly uses the CSC format. They are constructed analogously to each other, so only CSC will be introduced here.

A.3.1 *Compressed Sparse Column*

CSC works by only storing three one dimensional arrays instead of the full matrix. The first of these arrays stores the non-zero entries of the matrix. The second array stores the row indices of every entry in the non-zero value array. This, like the non-zero values array, is of length equal to the number of non-zeros (NNZ). The final array is known as the column pointer array and works by indicating at what index in the non-zero values array the next column of the matrix would begin at. For an $m \times n$ matrix, this array contains $n + 1$ entries.

As an example, consider the 5×5 matrix

$$M = \begin{pmatrix} 3 & 0 & 9 & 0 & 0 \\ 0 & 1 & 3 & 0 & 7 \\ 0 & 2 & 0 & 0 & 0 \\ 6 & 0 & 0 & 5 & 0 \\ 0 & 0 & 4 & 0 & 0 \end{pmatrix}. \tag{266}$$

The number of non zero entries in M is 9. Hence, the three storage vectors, of lengths 9, 9 and 6 respectively, can be constructed.

$$\text{Values} = [3 \;\; 6 \;\; 1 \;\; 2 \;\; 9 \;\; 3 \;\; 4 \;\; 5 \;\; 7] \tag{267}$$

$$\text{Row\_index} = [0 \;\; 3 \;\; 1 \;\; 2 \;\; 0 \;\; 1 \;\; 4 \;\; 3 \;\; 1] \tag{268}$$

$$\text{Col\_ptr} = [0 \;\; 2 \;\; 4 \;\; 7 \;\; 8 \;\; 9]. \tag{269}$$

Note that zero-based indexing is used here.

The difference between the consecutive entries in Col_ptr gives the number of non-zero entries in each column. For example, the number of entries in the second column of M is given by $4 - 2 = 2$. The values in this column correspond to values from the second to the fourth position in the Values array. CSC format is only more computationally efficient if

$$\text{NNZ} < \frac{(n(m-1)-1)}{2}. \tag{270}$$

B

NFESOLVE SUPPLEMENTARY MATERIAL

Here, we include the README.md file for the NFESOLVE repository in order to illustrate the steps needed to install the suite of code. The repository can be found at `https://github.com/UoN-Math-Neuro/NFESOLVE`.

B.1 README

NFESOLVE is a suite of differential equation solvers with a focus on the efficient computation of delay differential equations.

1. Make sure Armadillo is installed with an OpenBLAS backend (OpenBLAS isn't strictly necessary but provides optimised BLAS routines that will make code faster). See Armadillo's README.md for advice on this.
2. If you are wanting to run the parallel version of the code then you must have OpenMP 3.1 or later installed, as detailed in the Armadillo documentation.
3. Open terminal and cd to NFESOLVE directory. If you have OpenMP installed and wish to compile both the serial and parallel versions of NFESOLVE then just type '`make`'. To compile only the serial version type '`make NFESOLVE`', or for the parallel version type '`make NFESOLVE_PAR`'.

4. To include the NFESOLVE library in your project type '`#include "NFESOLVE.hpp"`' in the header definitions of your files.

5. See *Examples* folder for a variety of ODE and DDE example problems. For each problem a Makefile is supplied that follows the same template. It is recommended that you copy and edit the provided Makefiles to suit your own usage. It is important to make sure that the '`NFESOLVE_DIR`' variable in the Makefile is changed to the directory that you installed NFESOLVE in. The *DelayNFE_Example1* and *SparseDelayNFE_Example1* makefiles contain both serial and parallel versions which can be made individually by adding or removing the '`_PAR`' suffix to the make target when making.

BIBLIOGRAPHY

[1] Romesh G Abeysuriya, Jonathan Hadida, Stamatios N Sotiropoulos, Saad Jbabdi, Robert Becker, Benjamin AE Hunt, Matthew J Brookes, and Mark W Woolrich. A biophysical model of dynamic balancing of excitation and inhibition in fast oscillatory large-scale networks. *PLoS computational biology*, 14(2):e1006007, 2018.

[2] S Amari. Dynamics of pattern formation in lateral-inhibition type neural fields. *Biological Cybernetics*, 27(2):77–87, 1977.

[3] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

[4] G Arken, Hans J Weber, and FE Harris. Mathematical methods for physicists. *Academic Press, NY*, 1985.

[5] Ken Arnold and James Gosling. *The Java Programming Language*. Addison-Wesley, 1996.

[6] Fatihcan M Atay and Axel Hutt. Neural fields with distributed transmission speeds and long-range feedback delays. *SIAM Journal on Applied Dynamical Systems*, 5(4):670–698, 2006.

[7] K. E. Atkinson. *An introduction to numerical analysis*. Wiley, 1978.

[8] K. E. Atkinson. *The numerical solution of integral equations of the second kind*. Cambridge University Press, Cambridge, 1997.

[9] Kendall Atkinson, Weimin Han, and David E Stewart. *Numerical solution of ordinary differential equations*, volume 108. John Wiley & Sons, 2011.

[10] Daniele Avitabile. Projection methods for neural field equations. *arXiv preprint arXiv:2112.03244*, 2021.

[11] G. Azonos Beverido. Numerical computations of neural field models in triangulated meshes. Master's thesis, University of Nottingham, 2017.

[12] Peter J Basser, Sinisa Pajevic, Carlo Pierpaoli, Jeffrey Duda, and Akram Aldroubi. In vivo fiber tractography using DT-MRI data. *Magnetic resonance in medicine*, 44(4):625–632, 2000.

[13] John M Beggs. The criticality hypothesis: how local cortical networks might optimize information processing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 366(1864):329–343, 2008.

[14] John M Beggs. *The Cortex and the Critical Point: Understanding the Power of Emergence*. MIT Press, 2022.

[15] Timothy EJ Behrens, H Johansen-Berg, MW Woolrich, SM Smith, CAM Wheeler-Kingshott, PA Boulby, GJ Barker, EL Sillery, K Sheehan, O Ciccarelli, et al. Non-invasive mapping of connections between human thalamus and cortex using diffusion imaging. *Nature neuroscience*, 6(7):750–757, 2003.

[16] Alfredo Bellen and Marino Zennaro. *Numerical methods for delay differential equations*. Oxford University Press, 2013.

[17] Richard Ernest Bellman and Kenneth L Cooke. Differential-difference equations. 1963.

[18] Rani Ben-Yishai, R Lev Bar-Or, and Haim Sompolinsky. Theory of orientation tuning in visual cortex. *Proceedings of the National Academy of Sciences*, 92(9):3844–3848, 1995.

[19] L Susan Blackford, Antoine Petitet, Roldan Pozo, Karin Remington, R Clint Whaley, James Demmel, Jack Dongarra, Iain Duff, Sven Hammarling, Greg Henry, et al. An updated set of basic linear algebra subprograms (BLAS). *ACM Transactions on Mathematical Software*, 28(2):135–151, 2002.

[20] Przemyslaw Bogacki and Lawrence F Shampine. A 3 (2) pair of runge-kutta formulas. *Applied Mathematics Letters*, 2(4):321–325, 1989.

[21] Paul C Bressloff. Spatiotemporal dynamics of continuum neural fields. *Journal of Physics A: Mathematical and Theoretical*, 45(3):033001, 2011.

[22] Paul C Bressloff. Stochastic neural field theory. In *Neural Fields*, pages 235–268. Springer, 2014.

[23] Matthew J Brookes, Joanne R Hale, Johanna M Zumer, Claire M Stevenson, Susan T Francis, Gareth R Barnes, Julia P Owen, Peter G Morris, and Srikantan S Nagarajan. Measuring functional connectivity using MEG: methodology and comparison with fcMRI. *Neuroimage*, 56(3):1082–1104, 2011.

[24] John C Butcher. On Runge-Kutta processes of high order. *Journal of the Australian Mathematical Society*, 4(2):179–194, 1964.

[25] Áine Byrne. *Next generation neural activity models: bridging the gap between mesoscopic and microscopic brain scales*. PhD thesis, University of Nottingham, 2017.

[26] Áine Byrne, Daniele Avitabile, and Stephen Coombes. Next-generation neural field model: The evolution of synchrony within patterns and waves. *Physical Review E*, 99(1):012313, 2019.

[27] Áine Byrne, Matthew J Brookes, and Stephen Coombes. A mean field model for movement induced changes in the beta rhythm. *Journal of computational neuroscience*, 43(2):143–158, 2017.

[28] Áine Byrne, Reuben D O'Dea, Michael Forrester, James Ross, and Stephen Coombes. Next-generation neural mass and field modeling. *Journal of neurophysiology*, 123(2):726–742, 2020.

[29] Aine Byrne, James Ross, Rachel Nicks, and Stephen Coombes. Mean-field models for EEG/MEG: from oscillations to waves. *Brain Topography*, pages 1–18, 2021.

[30] F Ceschino. Evaluation de l'erreur par pas dans les problemes différentiels. *Chiffres*, 5:223–229, 1962.

[31] Francis Ceschino. Modification de la longueur du pas dans l'intégration numérique par les méthodes à pas liés. *Chiffres*, 2:101–106, 1961.

[32] P L Chebyshev. *Théorie des mécanismes connus sous le nom de parallélogrammes*. Imprimerie de l'Académie impériale des sciences, 1853.

[33] S. Coombes, P. beim Graben, R. Potthast, and J. Wright. *Neural fields: theory and applications*. Springer, 2014.

[34] Stephen Coombes. Waves, bumps, and patterns in neural field theories. *Biological Cybernetics*, 93(2):91–108, 2005.

[35] Stephen Coombes, Peter Beim Graben, and Roland Potthast. Tutorial on neural field theory. In *Neural Fields: Theory and Applications*, pages 1 – 43. Springer, 2014.

[36] Stephen Coombes and Áine Byrne. Next generation neural mass models. In *Nonlinear dynamics in computational neuroscience*, pages 1–16. Springer, 2019.

[37] Stephen Coombes and Markus R Owen. Evans functions for integral neural field equations with heaviside firing rate function. *SIAM Journal on Applied Dynamical Systems*, 3(4):574–600, 2004.

[38] Stephen Coombes and Helmut Schmidt. Neural fields with sigmoidal firing rates: approximate solutions. *Discrete and Continuous Dynamical Systems. Series S*, 2010.

[39] Stephen Coombes, Helmut Schmidt, and Ingo Bojak. Interface dynamics in planar neural field models. *The Journal of Mathematical Neuroscience*, 2(1):9, 2012.

[40] Stephen Coombes, NA Venkov, L Shiau, Ingo Bojak, David TJ Liley, and Carlo R Laing. Modeling electrocortical activity through improved local approximations of integral neural field equations. *Physical Review E*, 76(5):051901, 2007.

[41] Robert M Corless, Gaston H Gonnet, David EG Hare, David J Jeffrey, and Donald E Knuth. On the Lambert W function. *Advances in Computational Mathematics*, 5(1):329–359, 1996.

[42] Rodica Curtu and Bard Ermentrout. Pattern formation in a network of excitatory and inhibitory cells with adaptation. *SIAM Journal on Applied Dynamical Systems*, 3(3):191–231, 2004.

[43] Leonardo Dagum and Ramesh Menon. OpenMP: an industry standard API for shared-memory programming. *IEEE computational science and engineering*, 5(1):46–55, 1998.

[44] Gustavo Deco, Josephine Cruzat, Joana Cabral, Enzo Tagliazucchi, Helmut Laufs, Nikos K Logothetis, and Morten L Kringelbach. Awakening: Predicting external stimulation to force transitions between different brain states. *Proceedings of the National Academy of Sciences*, 116(36):18088–18097, 2019.

[45] Gustavo Deco, Viktor Jirsa, Anthony R McIntosh, Olaf Sporns, and Rolf Kötter. Key role of coupling, delay, and noise in resting brain fluctuations. *Proceedings of the National Academy of Sciences*, 106(25):10302–10307, 2009.

[46] Gustavo Deco and Viktor K Jirsa. Ongoing cortical activity at rest: criticality, multistability, and ghost attractors. *Journal of Neuroscience*, 32(10):3366–3375, 2012.

[47] Gustavo Deco, Morten L Kringelbach, Viktor K Jirsa, and Petra Ritter. The dynamics of resting fluctuations in the brain: metastability and its dynamical cortical core. *Scientific reports*, 7(1):3095, 2017.

[48] Murat Demirtaş, Joshua B Burt, Markus Helmer, Jie Lisa Ji, Brendan D Adkinson, Matthew F Glasser, David C Van Essen, Stamatios N Sotiropoulos, Alan Anticevic, and John D Murray. Hierarchical heterogeneity across human cortex shapes large-scale neural dynamics. *Neuron*, 101(6):1181–1194, 2019.

[49] S. Deng. Quadrature formulas in two dimensions, 2010. Math5172 - Finite Element Method.

[50] Rahul S Desikan, Florent Ségonne, Bruce Fischl, Brian T Quinn, Bradford C Dickerson, Deborah Blacker, Randy L Buckner, Anders M Dale, R Paul Maguire, Bradley T Hyman, et al. An automated labeling system for subdividing the human cerebral cortex on MRI scans into gyral based regions of interest. *Neuroimage*, 31(3):968–980, 2006.

[51] John E Desmedt and Guy Cheron. Central somatosensory conduction in man: neural generators and interpeak latencies of the far-field components recorded from neck and right or left scalp and earlobes. *Electroencephalography and clinical neurophysiology*, 50(5-6):382–403, 1980.

[52] John R Dormand and Peter J Prince. A family of embedded Runge-Kutta formulae. *Journal of Computational and Applied Mathematics*, 6(1):19–26, 1980.

[53] Gerald Ehrenstein and Harold Lecar. Electrically gated ionic channels in lipid bilayers. *Quarterly reviews of biophysics*, 10(1):1–34, 1977.

[54] James F Epperson. *An introduction to numerical methods and analysis*. John Wiley & Sons, 2013.

[55] Bard Ermentrout, Jonathan Rubin, and Remus Osan. Regular traveling waves in a one-dimensional network of theta neurons. *SIAM Journal on Applied Mathematics*, 62(4):1197–1221, 2002.

[56] G Bard Ermentrout, Stefanos E Folias, and Zachary P Kilpatrick. Spatiotemporal pattern formation in neural fields with linear adaptation. In *Neural Fields*, pages 119–151. Springer, 2014.

[57] G Bard Ermentrout and Nancy Kopell. Parabolic bursting in an excitable system coupled with a slow oscillation. *SIAM journal on applied mathematics*, 46(2):233–253, 1986.

[58] G Bard Ermentrout and David H Terman. *Mathematical foundations of neuroscience*, volume 35. Springer Science & Business Media, 2010.

[59] Jose M Esnaola-Acebes, Alex Roxin, Daniele Avitabile, and Ernest Montbrió. Synchrony-induced modes of oscillation of a neural field model. *Physical Review E*, 96(5):052407, 2017.

[60] L. Euler. *Institutionum calculi integralis*. Number v. 1 in Institutionum calculi integralis. imp. Acad. imp. Saènt., 1768.

[61] Grégory Faye and Olivier Faugeras. Some theoretical and numerical results for delayed neural field equations. *Physica D: Nonlinear Phenomena*, 239(9):561–578, 2010.

[62] Erwin Fehlberg. Low-order classical runge-kutta formulas with step-size control and their application to some heat transfer problems. *NASA technical report*, 1969.

[63] Stefanos E Folias. Nonlinear analysis of breathing pulses in a synaptically coupled neural network. *SIAM Journal on Applied Dynamical Systems*, 10(2):744–787, 2011.

[64] Stefanos E Folias and Paul C Bressloff. Breathing pulses in an excitatory neural network. *SIAM Journal on Applied Dynamical Systems*, 3(3):378–407, 2004.

[65] Michael Forrester, Jonathan J Crofts, Stamatios N Sotiropoulos, Stephen Coombes, and Reuben D O'Dea. The role of node dynamics in shaping emergent functional connectivity patterns in the brain. *Network Neuroscience*, 4(2):467–483, 2020.

[66] Michael Forrester, Sammy Petros, Yi Ming Lai, Reuben D O'Dea, Stamatios Sotiropoulos, and Stephen Coombes. Whole brain functional connectivity: insights from next generation neural mass modelling incorporating electrical synapses. *PLoS Computational Biology*, in preparation.

[67] Michael D Fox and Marcus E Raichle. Spontaneous fluctuations in brain activity observed with functional magnetic resonance imaging. *Nature reviews neuroscience*, 8(9):700–711, 2007.

[68] Ronald F Fox. Stochastic versions of the Hodgkin-Huxley equations. *Biophysical journal*, 72(5):2068–2074, 1997.

[69] Karl J Friston, Andrea Mechelli, Robert Turner, and Cathy J Price. Nonlinear responses in fMRI: the Balloon model, Volterra kernels, and other hemodynamics. *NeuroImage*, 12(4):466–477, 2000.

[70] Carl Friedrich Gauss. *Methodvs nova integralivm valores per approximationem inveniendi*. apvd Henricvm Dieterich, 1815.

[71] Matthew F Glasser, Timothy S Coalson, Emma C Robinson, Carl D Hacker, John Harwell, Essa Yacoub, Kamil Ugurbil, Jesper Andersson, Christian F Beckmann, Mark Jenkinson, et al. A multi-modal parcellation of human cerebral cortex. *Nature*, 536(7615):171–178, 2016.

[72] Matthew F Glasser, Stephen M Smith, Daniel S Marcus, Jesper LR Andersson, Edward J Auerbach, Timothy EJ Behrens, Timothy S Coalson, Michael P Harms, Mark Jenkinson, Steen Moeller, et al. The Human Connectome Project's neuroimaging approach. *Nature Neuroscience*, 19(9):1175, 2016.

[73] Matthew F Glasser, Stamatios N Sotiropoulos, J Anthony Wilson, Timothy S Coalson, Bruce Fischl, Jesper L Andersson, Junqian Xu, Saad Jbabdi, Matthew Webster, Jonathan R Polimeni, et al. The minimal preprocessing pipelines for the Human Connectome Project. *NeuroImage*, 80:105–124, 2013.

[74] Gary H Glover. Overview of functional magnetic resonance imaging. *Neurosurgery Clinics*, 22(2):133–139, 2011.

[75] Deanna J Greene, Christina N Lessov-Schlaggar, and Bradley L Schlaggar. Development of the brain's functional network architecture. In *Neurobiology of Language*, pages 399–406. Elsevier, 2016.

[76] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[77] E Hairer, S P Norsett, and G Wanner. *Solving Ordinary Differential Equations I*. Springer, Berlin, 1987.

[78] Jack K Hale. Functional differential equations. In *Analytic theory of differential equations*, pages 9–22. Springer, 1971.

[79] David Hansel and Haim Sompolinsky. Modeling feature selectivity in local cortical circuits. *Methods in Neuronal Modeling. From Synapses to Networks*, 01 1998.

[80] Martin Havlicek, Alard Roebroeck, Karl Friston, Anna Gardumi, Dimo Ivanov, and Kamil Uludag. Physiologically informed dynamic causal modeling of fMRI data. *Neuroimage*, 122:355–372, 2015.

[81] Suzana Herculano-Houzel. The human brain in numbers: a linearly scaled-up primate brain. *Frontiers in Human Neuroscience*, 3:31, 2009.

[82] Karl Heun. Neue methoden zur approximativen integration der differentialgleichungen einer unabhängigen veränderlichen. *Z. Math. Phys*, 45:23–38, 1900.

[83] Joerg F Hipp, David J Hawellek, Maurizio Corbetta, Markus Siegel, and Andreas K Engel. Large-scale cortical correlation structure of spontaneous oscillatory activity. *Nature neuroscience*, 15(6):884–890, 2012.

[84] Alan L Hodgkin and Andrew F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, 117(4):500–544, 1952.

[85] Allan L Hodgkin and Andrew F Huxley. The components of membrane conductance in the giant axon of loligo. *The Journal of physiology*, 116(4):473, 1952.

[86] Allan L Hodgkin and Andrew F Huxley. Currents carried by sodium and potassium ions through the membrane of the giant axon of loligo. *The Journal of physiology*, 116(4):449, 1952.

[87] Allan L Hodgkin and Andrew F Huxley. The dual effect of membrane potential on sodium conductance in the giant axon of loligo. *The Journal of physiology*, 116(4):497, 1952.

[88] Mary Kathleen Horn. Fourth- and fifth-order, scaled Runge-Kutta algorithms for treating dense output. *SIAM Journal on Numerical Analysis*, 20(3):558–568, 1983.

[89] G Evelyn Hutchinson. Circular causal systems in ecology. *Annals of the New York Academy of Sciences*, 50(4):221–246, 1948.

[90] Axel Hutt. *The study of neural oscillations by traversing scales in the brain*. PhD thesis, 2011.

[91] Axel Hutt, Michael Bestehorn, and Thomas Wennekers. Pattern formation in intracortical neuronal fields. *Network: Computation in Neural Systems*, 14(2):351–368, 2003.

[92] Axel Hutt and Nicolas Rougier. Numerical simulation scheme of one- and two dimensional neural fields involving space-dependent delays. In *Neural Fields*, pages 175–185. Springer, 2014.

[93] K J In't Hout. A new interpolation procedure for adapting runge-kutta methods to delay differential equations. *BIT Numerical Mathematics*, 32(4):634–649, 1992.

[94] Arieh Iserles. *A first course in the numerical analysis of differential equations*. Number 44. Cambridge University Press, 2009.

[95] Fudziah Ismail, Raed Ali Al-Khasawneh, Mohamed Suleiman, et al. Numerical treatment of delay differential equations by Runge-Kutta method using Hermite interpolation. *Matematika*, 18:79–90, 2002.

[96] ISO. ISO/IEC 14882:2011 Information technology — Programming languages — C++. *International Organization for Standardization, Geneva, Switzerland*, 27:59, 2012.

[97] Yasser Iturria-Medina, Erick Jorge Canales-Rodríguez, Lester Melie-García, Pedro A Valdés-Hernández, Eduardo Martínez-Montes, Yasser Alemán-Gómez, and JM Sánchez-Bornot. Characterizing brain anatomical connections using diffusion weighted MRI and graph theory. *Neuroimage*, 36(3):645–660, 2007.

[98] Eugene M Izhikevich. *Dynamical systems in neuroscience*. MIT press, 2007.

[99] Paul Jaccard. The distribution of the flora in the alpine zone. 1. *New phytologist*, 11(2):37–50, 1912.

[100] K R Jackson, Wayne H Enright, and T E Hull. A theoretical criterion for comparing runge–kutta formulas. *SIAM Journal on Numerical Analysis*, 15(3):618–641, 1978.

[101] Carl Gustav Jakob Jacobi. Ueber gauss neue methode, die werthe der integrale näherungsweise zu finden. 1826.

[102] JDG Jacobi. Ueber eine besondere gattung algebraischer functionen, die aus der entwicklung der function (1-2xz+ z2) 1/2 entstehen. 1827.

[103] Saad Jbabdi, Stamatios N Sotiropoulos, Suzanne N Haber, David C Van Essen, and Timothy E Behrens. Measuring macroscopic brain connections in vivo. *Nature Neuroscience*, 18(11):1546, 2015.

[104] Viktor K Jirsa and Hermann Haken. Field theory of electromagnetic brain activity. *Physical Review Letters*, 77(5):960, 1996.

[105] Viktor K Jirsa and Hermann Haken. A derivation of a macroscopic field theory of the brain from the quasi-microscopic neural dynamics. *Physica D: Nonlinear Phenomena*, 99(4):503–526, 1997.

[106] C. Ketelsen. Sparse data structures, 2015. Lecture notes for APPM 6640: Multigrid Methods. University of Colorado, Boulder.

[107] Peter E Kloeden and Eckhard Platen. Stochastic differential equations. In *Numerical Solution of Stochastic Differential Equations*, pages 103–160. Springer, 1992.

[108] Christof Koch. *Biophysics of computation: information processing in single neurons*. Oxford university press, 2004.

[109] Wilhelm Kutta. Beitrag zur näherungweisen integration totaler differentialgleichungen. *Z. Math. Phys.*, 46:435–453, 1901.

[110] C. R. Laing and A. Longtin. Noise-induced stabilization of bumps in systems with long-range spatial coupling. *Physica D Nonlinear Phenomena*, 160:149–172, December 2001.

[111] Carlo Laing and Stephen Coombes. The importance of different timings of excitatory and inhibitory pathways in neural field models. *Network: Computation in Neural Systems*, 17(2):151–172, 2006.

[112] Carlo R Laing. Spiral waves in nonlocal equations. *SIAM Journal on Applied Dynamical Systems*, 4(3):588–606, 2005.

[113] Carlo R Laing. PDE methods for two-dimensional neural fields. In *Neural Fields*, pages 153–173. Springer, 2014.

[114] Harold Lecar, Gerald Ehrenstein, and Ramon Latorre. Mechanism for channel gating in excitable bilayers. *Annals of the New York Academy of Sciences*, 264(1):304–313, 1975.

[115] Adrien Marie Legendre. Recherches sur l´ attraction des spheroides homogenes. 1785.

[116] Jason P Lerch, André JW van der Kouwe, Armin Raznahan, Tomáš Paus, Heidi Johansen-Berg, Karla L Miller, Stephen M Smith, Bruce Fischl, and Stamatios N Sotiropoulos. Studying neuroanatomy using MRI. *Nature neuroscience*, 20(3):314, 2017.

[117] P. M. Lima and E. Buckwar. Numerical solution of the neural field equation in the two-dimensional case. *SIAM Journal on Scientific Computing*, 37(6):B962–B979, 2015.

[118] T MacRobert. Spherical harmonics. an elementary treatise on harmonic functions. *Bull. Amer. Math. Soc*, 34:779–780, 1928.

[119] Daniel Marcus, John Harwell, Timothy Olsen, Michael Hodge, Matthew Glasser, Fred Prior, Mark Jenkinson, Timothy Laumann, Sandra Curtiss, and David Van Essen. Informatics and data mining tools and strategies for the human connectome project. *Frontiers in neuroinformatics*, 5:4, 2011.

[120] Rebecca Martin. *Collocation techniques for solving neural field models on complex cortical geometries*. PhD thesis, Nottingham Trent University, 2018.

[121] Gisiro Maruyama. Continuous markov processes and stochastic equations. *Rendiconti del Circolo Matematico di Palermo*, 4(1):48–90, 1955.

[122] Sam McKennoch, Thomas Voegtlin, and Linda Bushnell. Spike-timing error backpropagation in theta neuron networks. *Neural computation*, 21(1):9–45, 2009.

[123] R H Merson. An operational method for the study of integration processes. In *Proc. Symp. Data Processing*, pages 1–25. Weapons Research Establishment, Salisbury, S. Australia, 1957.

[124] Ernest Montbrió, Diego Pazó, and Alex Roxin. Macroscopic description for networks of spiking neurons. *Physical Review X*, 5(2):021028, 2015.

[125] Ernest Montbrió, Alex Roxin, et al. The role of fixed delays in neuronal rate models. *Centre de Recerca Matemàtica*, 2009.

[126] Florian Mormann, Klaus Lehnertz, Peter David, and Christian E Elger. Mean phase coherence as a measure for phase synchronization and its application to the EEG of epilepsy patients. *Physica D: Nonlinear Phenomena*, 144(3-4):358–369, 2000.

[127] Catherine Morris and Harold Lecar. Voltage oscillations in the barnacle giant muscle fiber. *Biophysical journal*, 35(1):193–213, 1981.

[128] Kenneth W Neves. Control of interpolatory error in retarded differential equations. *ACM Transactions on Mathematical Software*, 7(4):421–444, 1981.

[129] Paul L Nunez. The brain wave equation: a model for the EEG. *Mathematical Biosciences*, 21(3-4):279–297, 1974.

[130] Evert J Nyström. Über die praktische auflösung von integralgleichungen mit anwendungen auf randwertaufgaben. *Acta Mathematica*, 54(1):185–204, 1930.

[131] Edward Ott and Thomas M Antonsen. Low dimensional behavior of large systems of globally coupled oscillators. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 18(3):037113, 2008.

[132] Brynjulf Owren and Marino Zennaro. Continuous explicit runge-kutta methods. In *Institute of Mathematics and its Application Conference Series*, volume 39, pages 97–97. Oxford University Press, 1992.

[133] Christopher A H Paul. Developing a delay differential equation solver. *Applied Numerical Mathematics*, 9(3-5):403–414, 1992.

[134] Karl Pearson. VII. Note on regression and inheritance in the case of two parents. *proceedings of the royal society of London*, 58(347-352):240–242, 1895.

[135] David J Pinto and G Bard Ermentrout. Spatially structured activity in synaptically coupled neuronal networks: I. traveling fronts and pulses. *SIAM journal on Applied Mathematics*, 62(1):206–225, 2001.

[136] Sergio Pissanetzky. *Sparse Matrix Technology-electronic edition*. Academic Press, 1984.

[137] Dietmar Plenz, Tiago L Ribeiro, Stephanie R Miller, Patrick A Kells, Ali Vakili, and Elliott L Capek. Self-organized criticality in the brain. *Frontiers in Physics*, 9:639389, 2021.

[138] A. Quarteroni, R. Sacco, and F. Saleri. *Numerical Mathematics*, volume 37. Springer Science & Business Media, 2010.

[139] Anthony Ralston. Runge-Kutta methods with minimum error bounds. *Mathematics of computation*, 16(80):431–437, 1962.

[140] James Rankin, Daniele Avitabile, Javier Baladron, Gregory Faye, and David JB Lloyd. Continuation of localized coherent structures in nonlocal neural field equations. *SIAM Journal on Scientific Computing*, 36(1):B70–B93, 2014.

[141] F A Rihan, C Tunc, S H Saker, S Lakshmanan, and R Rakkiyappan. Applications of delay differential equations in biological systems. *Complexity*, 2018, 2018.

[142] Alex Roxin, Nicolas Brunel, and David Hansel. Role of delays in shaping spatiotemporal dynamics of neuronal activity in large networks. *Physical Review Letters*, 94(23):238103, 2005.

[143] Alex Roxin, Nicolas Brunel, and David Hansel. Rate models with delays and the dynamics of large networks of spiking neurons. *Progress of Theoretical Physics Supplement*, 161:68–85, 2006.

[144] Carl Runge. Über die numerische auflösung von differentialgleichungen. *Mathematische Annalen*, 46(2):167–178, 1895.

[145] Conrad Sanderson and Ryan Curtin. Armadillo: a template-based C++ library for linear algebra. *Journal of Open Source Software*, 1(2):26, 2016.

[146] Conrad Sanderson and Ryan Curtin. A user-friendly hybrid sparse matrix class in C++. In *International Congress on Mathematical Software*, pages 422–430. Springer, 2018.

[147] Conrad Sanderson and Ryan Curtin. An adaptive solver for systems of linear equations. In *2020 14th International Conference on Signal Processing and Communication Systems (ICSPCS)*, pages 1–6. IEEE, 2020.

[148] Paula Sanz Leon, Stuart A Knock, M Marmaduke Woodman, Lia Domide, Jochen Mersmann, Anthony R McIntosh, and Viktor Jirsa. The virtual brain: a simulator of primate brain network dynamics. *Frontiers in neuroinformatics*, 7:10, 2013.

[149] L F Shampine. Local extrapolation in the solution of ordinary differential equations. *Mathematics of Computation*, 27(121):91–97, 1973.

[150] L F Shampine and H A Watts. Global error estimation for ordinary differential equations. *ACM Transactions on Mathematical Software*, 2(2):172–186, 1976.

[151] Lawrence F Shampine. Interpolation for runge–kutta methods. *SIAM Journal on Numerical Analysis*, 22(5):1014–1027, 1985.

[152] Lawrence F Shampine and Laurent O Jay. Dense output. *Encyclopedia of Applied and Computational Mathematics; Springer: Berlin, Germany*, pages 339–345, 2015.

[153] Lawrence F Shampine and Mark W Reichelt. The MATLAB ODE suite. *SIAM Journal on Scientific Computing*, 18(1):1–22, 1997.

[154] Lawrence F Shampine and Skip Thompson. Solving DDEs in matlab. *Applied Numerical Mathematics*, 37(4):441–458, 2001.

[155] Lawrence. F Shampine and Herman A Watts. The art of writing a Runge-Kutta code. II. *Applied Mathematics and Computation*, 5(2):93–121, 1979.

[156] Greg Conradi Smith. *Cellular Biophysics and Modeling: A Primer on the Computational Biology of Excitable Cells*. Cambridge University Press, 2019.

[157] Hal L Smith. *An introduction to delay differential equations with applications to the life sciences*, volume 57. Springer New York, 2011.

[158] Stamatios N Sotiropoulos, Saad Jbabdi, Junqian Xu, Jesper L Andersson, Steen Moeller, Edward J Auerbach, Matthew F Glasser, Moises Hernandez, Guillermo Sapiro, Mark Jenkinson, et al. Advances in diffusion MRI acquisition and processing in the Human Connectome Project. *NeuroImage*, 80:125–143, 2013.

[159] Stamatios N Sotiropoulos and Andrew Zalesky. Building connectomes using diffusion MRI: why, how and but. *NMR in Biomedicine*, 2017.

[160] Larry Squire, Darwin Berg, Floyd E Bloom, Sascha Du Lac, Anirvan Ghosh, and Nicholas C Spitzer. *Fundamental neuroscience*. Academic press, 2012.

[161] Klaas E Stephan, Jeremie Mattout, Olivier David, and Karl J Friston. Models of functional neuroimaging data. *Current Medical Imaging*, 2(1):15–34, 2006.

[162] Bjarne Stroustrup. *The C++ Programming Language, First Edition*. Addison-Wesley, 1986.

[163] Endre Süli and David F. Mayers. *An Introduction to Numerical Analysis*. Cambridge University Press, 2003.

[164] Prejaas Tewarie, Romesh Abeysuriya, Áine Byrne, George C O'Neill, Stamatios N Sotiropoulos, Matthew J Brookes, and Stephen Coombes. How do spatially distinct frequency specific MEG networks emerge from one underlying structural connectome? the role of the structural eigenmodes. *NeuroImage*, 186:211–220, 2019.

[165] CORPORATE The MPI Forum. MPI: a message passing interface. In *Proceedings of the 1993 ACM/IEEE conference on Supercomputing*, pages 878–883, 1993.

[166] Thomas Trappenberg. *Fundamentals of Computational Neuroscience*. Oxford University Press, 2009.

[167] Kunichika Tsumoto, Hiroyuki Kitajima, Tetsuya Yoshinaga, Kazuyuki Aihara, and Hiroshi Kawakami. Bifurcations in Morris-Lecar neuron model. *Neurocomputing*, 69(4-6):293–316, 2006.

[168] Nathalie Tzourio-Mazoyer, Brigitte Landeau, Dimitri Papathanassiou, Fabrice Crivello, Olivier Etard, Nicolas Delcroix, Bernard Mazoyer, and Marc Joliot. Automated anatomical labeling of activations in spm using a macroscopic anatomical parcellation of the MNI MRI single-subject brain. *Neuroimage*, 15(1):273–289, 2002.

[169] Stephan Ulmer and Olav Jansen. *fMRI*. Springer, 2010.

[170] R Guido Van and F Drake. Python 3 reference manual. *Scotts Valley, CA: CreateSpace*, 10:1593511, 2009.

[171] David C Van Essen, Saad Jbabdi, Stamatios N Sotiropoulos, Charles Chen, Krikor Dikranian, Tim Coalson, John Harwell, Timothy EJ Behrens, and Matthew F Glasser. Mapping connections in humans and non-human primates: aspirations and challenges for diffusion imaging. In *Diffusion MRI (Second Edition)*, pages 337–358. Elsevier, 2014.

[172] David C Van Essen and Kamil Ugurbil. The future of the human connectome. *NeuroImage*, 62(2):1299–1310, 2012.

[173] David C Van Essen, Kamil Ugurbil, E Auerbach, D Barch, TEJ Behrens, R Bucholz, Acer Chang, Liyong Chen, Maurizio Corbetta, Sandra W Curtiss, et al. The Human Connectome Project: a data acquisition perspective. *NeuroImage*, 62(4):2222–2231, 2012.

[174] Stephan A van Gils, Sebastiaan G Janssens, Yu A Kuznetsov, and Sid Visser. On local bifurcations in neural field models with transmission delays. *Journal of mathematical biology*, 66(4):837–887, 2013.

[175] Nikola A Venkov. *Dynamics of neural field models*. PhD thesis, University of Nottingham, 2008.

[176] S Visser. Efficient evaluation of discretized neural fields with many delays (in preparation). 2017.

[177] Sid Visser, Rachel Nicks, Olivier Faugeras, and Stephen Coombes. Standing and travelling waves in a spherical brain model: the Nunez model revisited. *Physica D: Nonlinear Phenomena*, 349:27–45, 2017.

[178] David W Walker. Standards for message-passing in a distributed memory environment. Technical report, Oak Ridge National Lab., TN (United States), 1992.

[179] David W Walker and Jack J Dongarra. MPI: a standard message passing interface. *Supercomputer*, 12:56–68, 1996.

[180] Hugh R Wilson and Jack D Cowan. Excitatory and inhibitory interactions in localized populations of model neurons. *Biophysical Journal*, 12(1):1–24, 1972.

[181] Hugh R Wilson and Jack D Cowan. A mathematical theory of the functional dynamics of cortical and thalamic nervous tissue. *Kybernetik*, 13(2):55–80, 1973.

[182] Jens Wilting and Viola Priesemann. 25 years of criticality in neuroscience—established results, open controversies, novel concepts. *Current opinion in neurobiology*, 58:105–111, 2019.

[183] Zhang Xianyi, Wang Qian, and Zaheer Chothia. OpenBLAS. *URL: http://xianyi. github. io/OpenBLAS*, page 88, 2012.

[184] Tianye Zhai, Hong Gu, and Yihong Yang. Cox regression based modeling of functional connectivity and treatment outcome for relapse prediction and disease subtyping in substance use disorder. *Frontiers in Neuroscience*, 15:768602, 2021.

[185] J A Zonneveld. Automatic integration of ordinary differential equations. *Stichting Mathematisch Centrum. Rekenafdeling*, 743(63), 1963.