Dimension Reduction: Establishing Proofs and Extending Applications to the Cowan-Wilson Model

by

Alexander Robertus Van Solingen

to obtain the degree of Bachelor of Science
at the Delft University of Technology
to be defended publicly on Wednesday August 28, 2024 at 10:00

Student number:	5454158	
Thesis committee:	Dr. J. Dubbeldam	TU Delft, supervisor
	Dr. Y. Blanter	TU Delft, supervisor
	Dr. C. Kraaikamp	TU Delft
	Dr. T. Idema	TU Delft

Abstract

Models governed by systems of ordinary differential equations (ODEs) often produce complex and unpredictable behaviors. To address this, we can use dimension reduction techniques, which simplify these models, allowing for the retention of specific behaviors while greatly decreasing the cost of numerical solutions and, in some cases, enabling analytical derivations of sufficient conditions for the existence of nonzero fixed points of the model's ODEs. This thesis reviews the state-of-the-art reduction theories and extends the established proofs by Wu et al., by providing necessary assumptions, lemmas, and a novel proof of an important proposition used in their work. We additionally verify and confirm Wu et al.'s findings and predictions for a cooperative version of the Cowan-Wilson model, which describes a population of neurons' firing activity. We derived a one-dimensional reduction, inspired by Laurence et al., for a generalized Cowan-Wilson model, which we introduced in this thesis. Unlike the original, this generalized model can produce oscillatory behavior without external stimulus. A valuable finding is that the method of reduction does not depend on the specific form of the Cowan-Wilson function, allowing it to be applied to a broader class of nonlinear dynamics. Our reduction is able to predict system behavior effectively, *given* the network yields a unique reduction. However, the reduction parameter was not unique in approximately half of the networks studied, which saw a 66% increase in average error, suggesting these networks are inherently multi-dimensional. This opens the door for future research into the existence of a multi-dimensional reduction framework that could mitigate this discrepancy.

Table of Contents

1

Introduction

Everywhere you look, you will find systems of things that interact with one another: people and businesses in the Dutch economy, ants and their colony, neurons in your brain, animals in the Amazon rainforest, and so on. Some systems are made up of an enormous amount of components that interact with one another to create complex behavior, like neurons interacting to ‘make up’ your personality. As mathematicians and physicists, we seek to create models that can help us to understand such systems and especially predict changes in them. This could mean anything from determining the concentration of substrates in a gene regulatory network to the steady-state temperature of a system of objects. In many cases, ‘differential equations’ are suited to the task. That is, where for each component in the system (say each neuron’s activity in your brain), the change over time is related to other components (neurons) via some equation.

An example is the following general equation. Many systems obey an equation of this type, and that makes it interesting and valuable to study [1].

$$\frac{dx_i}{dt} = F(x_i) + \sum_{j=1}^{N} A_{ij} G(x_i, x_j). \tag{1.1}$$

Equation 1.1 describes the activity x_i of the ith component or *node* in a system of N *nodes*. The right-hand side is split into two parts, a self-interaction term $F(x_i)$ and a network-interaction term $\sum_{j=1}^{N} A_{ij} G(x_i, x_j)$, which is a weigh-ted sum over all nodes. Usually, $F(x_i)$ is negative representing self-decay, and $G(x_i, x_j)$ is nonnegative so that nodes generally increase their neighbors’ activities. Here $A_{ij} \in \mathbb{R}$ is a weight that qualitatively determines how much and in what way node j influences node i. In fact, these A_{ij} define a weighted directed graph on our N nodes, with the weight of the edge $j \to i$ equal to A_{ij}. We will refer to the $\mathbb{R}^{N\times N}$ matrix defined by these A_{ij} as A. Since the F and G in Equation 1.1 are the same for all nodes (no dependence on i or j), you would expect the graph structure to be quite indicative of the long-term activity of the system.

While this equation gives the basic structure of the models often considered, we will see the theory can handle slightly more general cases too. In the next section, we introduce such an example that will form the basis of application for the rest of the thesis.

Solving a system of N coupled differential equations may become difficult as N grows. However, using the network structure—the A_{ij}'s—it is possible to reduce the size, i.e., the dimension, of the system greatly. The general idea is to regard one, if not several, *weighted averages* of the nodes' activities: $\sum_{i=1}^{N} a_i x_i$. Then, if each weighted average is chosen carefully, the differential equations governing them are able to be well approximated by a simple, solvable system. Using this reduced system, we can study, for example, under what conditions the original system admits an equilibrium solution and how the underlying network affects these solutions. This reduction of dimension is the central theme of this thesis, and the bulk of this chapter contains a brief overview of what has been done in the Literature review. Finally, we give Physical justification of the research before covering some Notes on terminology and notation used throughout the thesis.

As for the structure of the rest of this thesis, it is split into two main parts: proofs and applications. Chapter 2 makes up the first category, where we build up a more rigorous setting for and cover the proofs behind some useful bounds introduced later in this chapter. In Chapter 3 we will then apply and benchmark these rigorous bounds on the cooperative Cowan-Wilson model, which leads us to Chapter 4, where we derive and study a dimension reduction for the complete noncooperative Cowan-Wilson model. After drawing conclusions in Chapter 5, there are a few appendices to be found. In Appendix A, we highlight the differences between cooperative and noncooperative systems with a few examples. In Appendix B, we give examples of the generalized Cowan-Wilson model and the reduction we derived for it. Finally, in Appendix C, we give the main pieces of code used to model and create graphs.

1.1 Introduction Cowan-Wilson model

An example of a model that fits into the framework is the Cowan-Wilson neuron model in computational neuroscience [2]. The model gives equations for the *proportion of a subpopulation of neurons* that is 'firing' at any given time. The original paper considered two subpopulations, and it was able to generate hysteresis and (damped) oscillatory behavior, which the authors related to physiological experiments. We regard a generalized version on N subpopulations, with x_i the proportion of firing neurons of the ith subpopulation. The x_i then satisfy the following differential equation:

$$\frac{dx_i}{dt} = -x_i + Z\left(\sum_{j=1}^{N} A_{ij}x_j + p_i\right), \tag{1.2}$$

where Z is a 'smoothed-out' step function, A_{ij} are the mean connections between subpopulations and p_i is a constant that represents the external stimulus that the ith subpopulation receives. The equation qualitatively has a similar form to (1.1), where x_i naturally decays over time, if it were not for the connection to other neurons. The total weighted incoming activity is passed through a function Z that maps its argument between 0 and 1. We will explain this function and the rest of the model in detail in Section 4.1, and we will use a restricted version to test a reduction in Chapter 3.

1.2 Literature review

Much thought has already been put into the problem of dimension reduction. In this thesis we mainly regard the theoretical work done by Laurance *et al.* in 2019 [1] and Wu *et al.* in 2023 [3], which builds upon the pioneering work done by Gao *et al.* in 2016 [4].

1.2.1 Assumption: cooperative networks

An important assumption made in all mentioned papers is that networks are cooperative, which means $A_{ij} \geq 0$ for all connections. Note that in Cowan and Wilson's paper mentioned above, for example, they explicitly mention that only modeling excitatory neurons, which stimulate other neurons' activity, is insufficient; inhibitory neurons, which inhibit other neurons' activity, are crucial to accurately modeling the brain. We model inhibitory interactions by setting the corresponding $A_{ij} < 0$. Thus, assuming $A_{ij} \geq 0$ in neural models means we are missing inhibitory neurons: a severe limitation. In Appendix A, give examples of the differences for an epidemic model.

That said, there are other models that can be fully described by cooperative networks, such as those corresponding to a system of plants and their pollinators as studied in [5]. Thus there is enough physical reason to study systems subject to this limitation, besides mathematical ease.

1.2.2 Gao *et al.*

The idea behind the dimension reduction explained in the Introduction, originates in a paper by Gao *et al.* in 2016 [4]. In it, the authors consider Equation 1.1, and define the following weighted average:

$$R := \frac{\mathbf{1}^T A \boldsymbol{x}}{\mathbf{1}^T A \mathbf{1}}, \tag{1.3}$$

for A as above. To gain some intuition on this definition, recall the system defines a weighted, directed graph with A_{ij} the weight of the $j \to i$ edge. Thus, the j'th column of A represents the weights of the outgoing connections of node j, and $\mathbf{1}^T A$ is a vector $\boldsymbol{s}_{\text{out}}^T$ with the outgoing degree (sum of outgoing weights) of each node. Thus,

$$R = \frac{\sum_{i=1}^N s_{\text{out,i}} x_i}{\sum_{i=1}^N s_{\text{out,i}}}, \tag{1.4}$$

so we weigh a node's activity by its out-degree. This means nodes that have a strong influence on other nodes contribute more to R. Finally, after making several approximations, including an assumption on the degree distribution of the network, the authors find R can be described by:

$$\frac{dR}{dt} \approx F(R) + \beta_{\text{eff}} G(R, R), \tag{1.5}$$

with $\beta_{\text{eff}} = (\mathbf{1}^T A^2 \mathbf{1})/(\mathbf{1}^T A \mathbf{1})$ a network parameter. We will see how this relates to the next reduction.

1.2.3 Laurence *et al.*

In 2019, Laurence *et al.* set out to systematize this dimension reduction [1]. Starting with the one-dimensional case, the authors define R more generally as,

$$R = \sum_{i=1}^{N} a_i x_i = \boldsymbol{a}^T \boldsymbol{x}, \tag{1.6}$$

with $\boldsymbol{a}$ a to-be-determined weight vector. When plugging R into Equation 1.1, and performing two Taylor expansions, about $F(R)$ and about $G(\beta R, R)$, one finds up to error in $\mathcal{O}[(x_i - R)^2]$,

$$\frac{dR}{dt} = F(R) + \alpha G(\beta R, R), \tag{1.7}$$

where cancellation of $\mathcal{O}[x_i - R]$ terms forces α to satisfy the following eigenproblem:

$$A^T \boldsymbol{a} = \alpha \boldsymbol{a}. \tag{1.8}$$

Here $\boldsymbol{a}$ is chosen as the positive eigenvector with the largest real eigenvalue (the existence of which is guaranteed by the Perron-Frobenius theorem [6]). The authors maximize α because they want the $G(\beta R, R)$ term to significantly influence dR/dt. On the other hand, if α were relatively small, we would expect the dropped terms in $\mathcal{O}[(x_i - R)^2]$ to become more significant. Furthermore, β is given by:

$$\beta = \frac{1}{\alpha} \frac{\boldsymbol{a}^T K \boldsymbol{a}}{\boldsymbol{a}^T \boldsymbol{a}}, \tag{1.9}$$

where K is a diagonal matrix with elements $K_{ii} = \sum_j A_{ij}$, i.e., the in-degree of node i. We see that, ideally, $\boldsymbol{a}$ is also an eigenvector of K with eigenvalue $\alpha\beta$. Since this is not generally possible, β is taken as the least squares solution. Then, the authors showed that the reduction above (1.2.2) is a special case of this one[1]. However, they did not stop there.

By defining n weighted averages, one may be able to capture more of the network structure in the reduction: $R_k := \boldsymbol{a}_k^T \boldsymbol{x}$ for $k = 1, 2, ..., n$. This time we

[1]'Special case' because Gao *et al.* make further limiting assumptions on the degree distribution of the network.

expand around $G(\beta_k R_k, R_{k+1})$ and $F(R_k)$ aiming to derive cyclic equations for the R_k:

$$\frac{dR_k}{dt} = \begin{cases} F(R_k) + \alpha_k G(\beta_k R_k, R_{k+1}) & k < n, \\ F(R_k) + \alpha_k G(\beta_k R_k, R_1) & k = n, \end{cases} \tag{1.10}$$

which we attain when the $\boldsymbol{a}_k$ satisfy slightly different conditions:

$$\boldsymbol{a}_{k+1} = \frac{A^T \boldsymbol{a}_k}{\alpha_k} \tag{1.11}$$

where $\alpha_k := \boldsymbol{a}^T A \mathbf{1}$, and we require $\boldsymbol{a}_{k+n} = \boldsymbol{a}_k$ (cyclic). Finding a suitable $\boldsymbol{a}$ becomes a little more involved, so we refer to the paper [1] for details. We remark that the multidimensional reduction is in principle applicable only when there are multiple eigenvalues equal to the spectral radius, i.e., multiple (distinct) dominant eigenvalues.

1.2.4 Wu *et al.*

Finally, as recently as 2023, Wu *et al.* took another approach [3]. They argue the progress made thus far is only approximating: in the case of Laurence *et al.* a first order Taylor expansion, which may quickly lose accuracy for nonlinear models. Instead, the authors focused on finding rigorous bounds for when the system has a nonzero fixed point. If a system has such a nonzero fixed point, we say it *survives*, lending the terminology from ecological models. If it does not, we say it *collapses*.

The authors start with a more general dynamic equation,

$$\frac{dx_i}{dt} = H\left(x_i, \sum_{j=1}^{N} A_{ij} G(x_i, x_j)\right), \tag{1.12}$$

with a list of constraints on H that we come back to in Section 2.1. Note Equation 1.1 is a special case of (1.12), retrieved by defining $H(x_i, u_i) = F(x_i) + u_i$. The function H usually contains some parameters, α, β, γ, etc. which we collectively refer to as the *dynamics* of the model (as opposed to the *network* imposed parameters, A_{ij}).

The main idea of the paper is to regard a 'symmetric network', "in which all nodes are equal to one another" (*p.* 2 of the paper)[2]. That is, consider a simplification of Equation 1.12, where x_i and x_j are replaced by x (one node):

$$\frac{dx}{dt} = H(x, \lambda G(x, x), \alpha, \beta, \gamma, ...), \tag{1.13}$$

where $\lambda = \sum_j A_{ij}$, the incoming degree (equal for all nodes), and we include the dynamic parameters $\alpha, \beta, \gamma, ...$ to emphasize dependence on them. Now,

[2]This is an admittedly vague definition, and there is indeed an ambiguity in it that is explored later, in the Intermezzo on 'symmetric networks'.

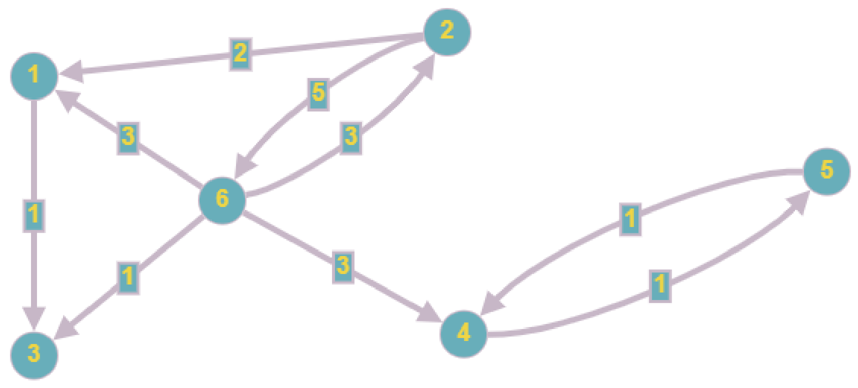


Figure 1.1: A weighted, directed graph on 6 vertices. $k_{\max} = 3$, found by taking the induced subgraph on vertices $\{1, 2, 6\}$. Including vertex 3, makes the induced subgraph 2-core. The entire network is 1-core.

studying stable solutions of a one-dimensional ODE like (1.13) can usually be easily done numerically, if not analytically. What is left is to relate equation (1.13) to the original network's dynamic equation.

Given dynamic parameters $\alpha, \beta, \gamma, \ldots$, the authors define λ^* as the smallest $\lambda \geq 0$ such that (1.13) has a nonzero fixed point x^*. We can interpret λ^* as the effective difficulty of survival of the set of parameters. It can be calculated in terms of the dynamic parameters by setting (1.13) equal to zero.

To build the relation, they prove that if (1.13) has a solution for $\lambda = k_{\max}(A)$, then the main Equation 1.12 has a solution ($k_{\max}(A)$ defined below). Conversely, if Equation 1.12 has a solution, then so does (1.13) with $\lambda = \rho(A)$ ($\rho(A)$ defined below). This translates, respectively, to that $\lambda^* \leq k_{\max}(A)$ implies the original system survives and $\rho(A) \leq \lambda^*$ implies the original system collapses. In other words, the so-called 'tipping points', where we go from survival to collapse, will occur for $\alpha, \beta, \gamma, \ldots$ such that $\lambda^* \in [k_{\max}(A), \rho(A)]$.

Here,

Definition 1. *$k_{max}(A)$ is the maximum so-called k-coreness of the network, A. That is, for the network A, $k_{max}(A)$ is the largest number $d \in \mathbb{R}$ such that there is a selection of nodes such that in the induced subgraph of these nodes each node has a total incoming degree greater than or equal to d.*

In general we refer to such a subgraph as being d-core. In Fig. 1.1, a simple network is illustrated as an example. Its $k_{\max}$ can be read off at 3.

Additionally,

Definition 2. *$\rho(A)$ is defined as the largest eigenvalue of the adjacency matrix of the network A.*

For in the network shown in Fig. 1.1, $\rho(A) \approx 3.87$. We will show later that $k_{\max}(A) \leq \rho(A)$ generally holds, so that the bounds are well defined.

In closing, notice the similarity in (1.13) to the Gao reduction (1.5) and Laurence one-dimensional reduction (1.7). Wu *et al.* have essentially given conditions for the latter reductions to predict with certainty the survival and collapse of the original system.

With Wu *et al.*ś framework, it is possible to predict whether networks survive, given the dynamics governing each node, by regarding the solution space of a one-dimensional simplified ODE. In the next chapter, we will cover and discus the authors' proof of the two bounds which make this reduction work.

1.3 Physical justification of the research

As described in the introduction, there are many models which fall into the framework we consider in this thesis. These include more 'classical' physical systems, such as the Kuramoto model of oscillators (which has applications all through science, including arrays of Josephson junctions [7]), or a model of the temperature of coupled objects subject to Newton's law of cooling Equation 1.1.

The topic of this thesis is dimension reduction. This has the direct application of being able to easily calculate (and thus predict) a weighted average of a large system's activities at any point in time, which can be useful in the mentioned Kuramoto or temperature models. A consequence is that reductions can be used to predict an average *in steady state*, in particular determining whether a system does or does not have a nonzero steady state. If the system was a population model, another typical example fitting in the framework, this is equivalent to all (or some) species surviving—very important information. Furthermore, reduced systems lend themselves to being analytically solvable, solutions of which can be used to make statements about (larger) systems in general. For example, we can derive conditions on the parameters of a reduction such that its governing equations have a nonzero solution and then conclude this gives approximate conditions such that a general network has a nonzero solution (because the reduction parameters depend on the network and dynamical parameters).

In this thesis, we study yet another model: that of a population of neurons' firing activity. We will apply known theory to a restricted version and then reintroduce it in its physical context. For this full model, we derive a reduction and attempt to draw physical conclusions based thereon.

1.4 Notes on terminology and notation

- Throughout this entire thesis when we write inequalities for vectors and matrices, we mean taking the inequality element-wise. Special cases are 'nonzero', where we mean that there is at least one entry that is nonzero, and 'positive', where we mean nonnegative and nonzero.

- We refer to the usual l^2 norm on $\mathbb{C}^n$ as $|\cdot|_2$.
- We use * to refer to the complex conjugate. Sometimes we also use it to label a particular solution or variable. It is clear from context when we mean which one.
- Boldface variables refer to a vector, e.g. $\boldsymbol{v} \in \mathbb{R}^n$
- $\mathbb{1}_{x \in A}$ refers to an indicator function for the set A. We do not refer to it as a function of x (i.e. with the suffix '(x)') as this is implied by x being the variable in subscript.
- We use $\mathcal{O}(\cdot)$ to describe the well-known big-O notation of the asymptotic behavior of functions.

2

Rigorous Bounds Proof and Symmetric Networks

In this chapter we cover the proofs from Wu et al.'s paper that provides rigorous bounds for network collapse [3]. After proving a preliminary result we examine a slight ambiguity as to what a 'symmetric network' is, and explore its details. To complete the rigorous bounds proof, we need an additional proposition, which we present an elementary proof for, although there is already existing theory that deals with the problem, but which we cover very briefly.

As a reminder, the rigorous bounds, explained in the previous section are that

$$\lambda^* \leq k_{\max}(A) \text{ implies the network always has a positive solution,} \tag{2.1}$$
$$\rho(A) \leq \lambda^* \text{ implies 0 is the only stable solution,} \tag{2.2}$$

where λ^* is an indicator of the 'difficulty' of the system's dynamics.

A couple points on notation and terminology: We will drop out the '(A)' behind $k_{\max}$ and ρ as we are always referring to the same adjacency matrix: that of the main system of N nodes. We use 'fixed point', 'equilibrium point' and 'solution' interchangeably. All refer to a point $\boldsymbol{x}_0$ such that $d\boldsymbol{x}/dt(\boldsymbol{x}_0) = \mathbf{0}$[1]. Only seldomly does 'solution' refer to the curve $\boldsymbol{x}(t)$ that satisfies a differential equation, in which case it will be clear from context. We say a system 'survives' if there it has a nonzero fixed point, and 'collapses' otherwise. We will see that we always demand zero to be a solution. Thus, a system collapsing usually means that zero is the only stable point, so all activities eventually fall to zero.

[1]The underlying assumption here is that we only regard autonomous systems, those where the function for $d\boldsymbol{x}/dt$ is not a function of time.

2.1 Rigorous bounds assumptions

Let us first line out the assumptions the authors (Wu *et al.*) make on the dynamics, that is on Equation 1.12:

Assumption 1. *$H(\boldsymbol{x} = \mathbf{0}) = 0$ and $\lim_{||\boldsymbol{x}|| \to \infty} H(\boldsymbol{x}) < 0$: $\mathbf{0}$ must be a fixed point, i.e. the network is able to collapse, and it will not grow out to infinity*[2].

Assumption 2. *$H\left(x_i > 0, \sum_{j=1}^{N} A_{ij} G(x_i, x_j) = 0\right) < 0$: a node's activity will decay if it receives no incoming activity.*

Assumption 3. *Writing $H = H(x_i, u_i)$, $\partial H / \partial u_i \geq 0$: incoming activity cannot decrease a node's activity.*

Assumption 4. *$\partial G / \partial x_j \geq 0, G(x_i, 0) = 0$: support from neighbors increases with the increase of their own activity, and collapsed neighbors offer no support.*

The first two are generally true for a real network, whereas the latter two can be considered restrictions, forcing the network to be cooperative in dynamics as well as in its network.

We have the following useful immediate consequence of Asm. 4:

Lemma 1. *G is nonnegative: $G(x_i, x_j) \geq 0$ for $x_j \geq 0$.*

Proof. Because $\partial G / \partial x_j \geq 0$, $G(x_i, x_j) \geq G(x_i, 0) = 0$. ■

As a reminder, we assume throughout this chapter that the network is cooperative, meaning $A_{ij} \geq 0$ for all i, j.

2.2 Well-definedness of the bound

Firstly, we show the pair of bounds is well-defined, and refer to Def. 1 and Def. 2 for the definitions the parameters. We achieve this by regarding the Rayleigh quotient with a $k_{\max}$-achieving vector.

Proposition 1. *k_{max} is less than or equal to ρ, thus the rigorous bounds are always well-defined.*

Proof. Given a set S of nodes such that the induced subgraph on S is $k_{\max}$-core, define the vector v by $v_i = \mathbb{1}_{i \in S}$. Then, the so-called Rayleigh quotient in v is given by (the leftmost quotient):

$$\frac{v^T A v}{v^T v} = \frac{\sum_{i=1}^{N} v_i (Av)_i}{|S|} = \frac{\sum_{i \in S} (Av)_i}{|S|} \geq \frac{\sum_{i \in S} k_{\max}}{|S|} = k_{\max}, \tag{2.3}$$

[2]The latter assumption is actually too strong, and does not apply to most networks because this limit may not exist e.g. for the gene regulatory network model regarded in the paper [3]. Depending on the path P you take $\boldsymbol{x}$ to infinity, the correct assumption is that the limit of H_i is negative if P takes $x_i \to \infty$, where $H_i = dx_i/dt$. This way we at least know a node's own growth will eventually cause its activity to decrease, and we cannot ask for more because, for example, there are paths to infinity for which the SIS model's derivative grows to infinity (where the SIS model is another model regarded in the paper).

where the inequality follows by definition of k-coreness of the induced subgraph on S. Linear algebra tells us the Rayleigh quotient achieves a maximum at ρ [6], thus $k_{\max} \leq \rho$. ∎

2.3 Survival: $\lambda^* \leq k_{\max}$

Recall the inequality $\lambda^* \leq k_{\max}$ means by definition of λ^* that a symmetric network with $\lambda = k_{\max}$ survives. Thus, the bound is equivalent to proving that if said symmetric network survives, then the original network survives. Because of the nature of the k-core of a graph, this bound is simple—on the surface—to prove.

Proposition 2. *When $\lambda^* \leq k_{max}$, the original network has a nonzero fixed point.*

Proof. (Wu et al.) Suppose a symmetric network with in-degree $k_{\max}$ has a positive solution. By definition of $k_{\max}$, we can remove nodes and reduce weights such that each node in the network has in-degree $k_{\max}$. Now, we assumed such a symmetric network has a positive solution, $\boldsymbol{x}^*$.

The authors then state that this solution is equal for all nodes i: $x_i^* = x^*$, and then that—given this fixed point of the system with reduced network—the original system must have a solution for which the nonzero activities are no less than x^*, thus finishing the proof. ∎

These final two statements are critical in the proof, but the authors do not supply any arguments why they should be true. We investigate the first statement in the following intermezzo, and return to the second one in Section 2.5

2.4 Intermezzo on 'symmetric networks'

As explained in Section 1.2.4, the main idea of the paper is to regard a 'symmetric network', "in which all nodes are identical to each other," as the authors define it. Finding a fixed point in this system is easier and then can be used to prove (non)existence of a solution in the original system. However, we find that the mentioned non-technical definition naturally leads to two different interpretations.

2.4.1 Interpretations

There are two interpretations of what constitutes a 'symmetric network':

1. One-node interpretation: A network in which each node has the same degree distribution *and* activity or, equivalently, a one-node network described by:
$$\frac{dx}{dt} = H\left(x, \lambda G(x,x)\right), \tag{2.4}$$
where λ represents the in-degree.

2. Network interpretation: A network of N nodes in which each node has the same total incoming weight, thus, the system is described by:

$$\frac{dx_i}{dt} = H\left(x_i, \sum_{j=1}^{N} A_{ij} G(x_i, x_j)\right), \tag{2.5}$$

with $\lambda = \sum_j A_{ij}$ total in-degree, equal for all nodes.

It is clear that if (2.4) has a fixed point x^*, then $x_i = x^*$ for all i is a solution of (2.5). In the proof of the paper, however, the authors mistakenly use the converse: they assume[3] that a symmetric system in the sense of (2.5), acquired by removing nodes and reducing weights, has a solution *and then* conclude that the solution is equal for all nodes. We show by counterexample, however, that this converse does not generally hold.

Fortunately, the converse is not required. Recall we define λ^* as the smallest value such that Equation 1.13 has a nonzero fixed point. Thus, when we suppose $\lambda^* \leq k_{\max}$, it means (2.4) has a nonzero fixed point, implying that (2.5) has a symmetric nonzero solution, which is what the authors actually use in the proof.

Thus, the first step of the proof is justified. Before moving to the second, we present and discuss a counterexample as promised.

2.4.2 Counterexample

Consider the following dynamic system in the form of Equation 1.1 and plot of its decay function:

$$F(x_i) = -x_i^3 + 3x_i^2 - \frac{5}{2}x_i, \tag{2.6}$$

$$G(x_i, x_j) = \frac{1}{2}x_j, \tag{2.7}$$

$$A = \begin{pmatrix} 0 & 1/2 & 0 & 1/2 \\ 1/2 & 0 & 1/2 & 0 \\ 0 & 1/2 & 0 & 1/2 \\ 1/2 & 0 & 1/2 & 0 \end{pmatrix}. \tag{2.8}$$

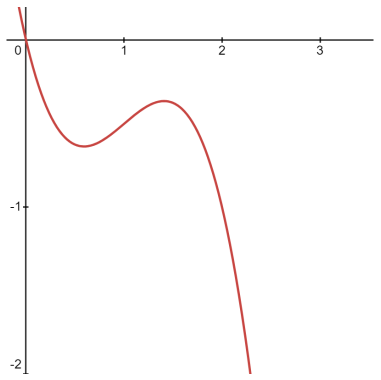


Figure 2.1: Plot of $F(x_i)$, a negative and non-injective decay function.

It is straightforward to verify that $\boldsymbol{x} = (1 + 1/\sqrt{2}, 1, 1 - 1/\sqrt{2}, 1)^T$ is a non-symmetric fixed point of this system (although it may not be stable), and that

[3]You can verify this for yourself here, under ‘Condition 2’.

all four assumptions (see Section 2.1) are satisfied. On the other hand, every node has incoming degree equal to one ($\lambda = 1$) meaning the system is symmetric in the sense of the network interpretation. Because the selected fixed point is not symmetric, this shows that a solution of a network-interpretation-symmetric system is not necessarily equal in all nodes. Here, the trick was to use a non-injective decay function, which here gave us $F(1+1/\sqrt{2}) = F(1-1/\sqrt{2}) = -\frac{1}{2}$.

The underlying network is, in fact, a 4-cycle with all edge weights set at $1/2$. We can generalize this counterexample to a $4n$-cycle network with weights at $1/2$, by repeating the 4-cycle solution vector. This works because in a cycle each node only 'sees' its two immediate neighbors, thus the exact same calculations as for the 4-cycle yield the repeated vector is a stable point.

2.4.3 A symmetric solution exists

Even though the above counterexample has nonsymmetric solutions, there still are solutions that are equal for all nodes (e.g. $(1,1,1,1)^T$). Thus, the question arises whether it is true that, given there is a positive solution to a symmetric network, *there exists* a positive solution that is equal for all nodes. It turns out this is the case, but to prove it we need a short lemma, which we will use later as well, as it is a defining quality of the systems studied.

Lemma 2. *Writing $H_i(\boldsymbol{x}) := H(x_i, \sum_j A_{ij} G(x_i, x_j)) = H(x_i, u_i)$, H_i is increasing in x_j for all $j \neq i$.*

Proof. Using the chain rule:

$$\frac{\partial H_i}{\partial x_j} = \frac{\partial H_i}{\partial u_i} \frac{\partial u_i}{\partial G(x_i, x_j)} \frac{\partial G(x_i, x_j)}{\partial x_j} = \frac{\partial H_i}{\partial u_i} A_{ij} \frac{\partial G(x_i, x_j)}{\partial x_j} \geq 0, \tag{2.9}$$

where the final inequality holds by Asm. 3 & 4 and the cooperative network assumption. ■

Proposition 3. *If a symmetric system in the sense of the network interpretation has a nonzero equilibrium point $\boldsymbol{x}$, then it will also have a nonzero equilibrium point $\boldsymbol{y}$ such that $y_i = y$ is equal for all nodes, i.e., the symmetric system in the sense of the one-node interpretation has a nonzero solution.*

Proof. Suppose a symmetric system in the sense of the network interpretation with incoming degree λ has a nonzero solution: $\boldsymbol{x} = (x_1, ..., x_N)^T$. Let,

$$x_m = \max\{x_i : i = 1, ..., N\}, \tag{2.10}$$

for m the index of a maximal node. Note that $x_m \neq 0$ because $\boldsymbol{x}$ is nonzero. Then we find:

$$0 = H\left(x_m, \sum_{j=1}^{N} A_{mj} G(x_m, x_j)\right) \leq H\left(x_m, G(x_m, x_m) \sum_{j=1}^{N} A_{mj}\right) \tag{2.11}$$

$$= H\left(x_m, \lambda G(x_m, x_m)\right), \tag{2.12}$$

where the first equality is by definition of $\boldsymbol{x}$ and the inequality holds by Lemma 2. So we find that $H(x, \lambda G(x, x))$ is nonnegative at x_m. Now, because H becomes negative in the limit to infinity (by Asm. 1), the intermediate value theorem gives us that there is a value $x^* \geq x_m > 0$ such that $H(x^*, \lambda G(x^*, x^*) = 0$. ■

Because we know the converse is already true, we find in fact that the *existence* of a solution in a one-node interpretation is equivalent to the *existence* of a solution in a network interpretation.

2.5 Survival: $\lambda^* \leq k_{\max}$ cont.

We have shown that there may be more to the solution of a symmetric network than meets the eye, but there is one more step remaining in the proof of the survival bound. Namely, we now have a solution $\boldsymbol{x}^*$ of the $k_{\max}$-symmetric reduced network, but now we need to find a solution of the larger original network. Recall that to go from the symmetric to the original network, we have to add back in nodes and their connections and re-increase some weights. However, if we regard the removal of a node as setting all its connections to zero, adding a node back in constitutes re-increasing weights. Thus, it suffices to show that if the system has a positive solution before increasing some A_{ij}, it will have one afterward as well.

The crux of the proof is based on Lemma 2. That is, one node's increasing activity cannot decrease its neighboring node's activity. Thus, intuitively, when we increase a weight, the original nodes' activities will not decrease. We find the entire network's activity does not decrease, but as it cannot grow to infinity, it approaches a stable point somewhere along the way.

To complete the proof in its current state, a few extra assumptions have to be made on the dynamics and network of the system.

Assumption 5. *G and H are strictly monotone with respect to their second input: $\partial G / \partial x_j > 0$ and $\partial H_i / \partial u_i > 0$.*

Assumption 6. *A is symmetric: $A_{ij} = A_{ji}$.*

We remark that all systems exhibited by Wu et al. and Laurence et al. satisfy Asm. 5. Thus, only Asm. 6 is restrictive, although, in some cases symmetry is a natural property of the system being analyzed, for example if one models a system of oscillators with equal masses or a highly contagious disease (where the probability of infection only depends on if two people come into contact).

A final important remark to be made is that the main step in the following proof—showing that the solution $\boldsymbol{x}(t)$ is nondecreasing in time—can be proven using theory on quasimonotone functions, *without the additional assumptions we make*. The necessary theory is explained in [8], and in particular we use Theorem 4, where $g = H$ is a so-called quasimonotone function because for each i, H_i is increasing in all $j \neq i$ (Lemma 2). We present our weaker statement as a proposition and give the proof below.

Proposition 4. *Given a network A and a vector $\boldsymbol{x}_0$ s.t. $\boldsymbol{x}_0$ is a nonzero fixed point to ODE Equation 1.12 with network A; the ODE Equation 1.12 with network $A + D$, for $D_{ij} = A_{ij} + \mathbb{1}_{\{(i,j)=(m,n)\}}\Delta$, has a nonzero fixed point $\boldsymbol{y}_0 \geq \boldsymbol{x}_0$. for any $\Delta \geq 0$ and pair of indices $m = 1, ..., N$ and $n = 1, ..., N$.*

Proof. Suppose a network A on N nodes adhering to Equation 1.12 has a solution $\boldsymbol{x}^*$, that is, for each i:

$$H_i\left(x_i^*, \sum_{j=1}^{N} A_{ij} G(x_i^*, x_j^*)\right) = 0. \tag{2.13}$$

Select a connection (m, n) and increase and replace $A_{mn} \mapsto A_{mn} + \Delta$ for $\Delta > 0$. Regard the new initial value problem with initial conditions at the old equilibrium point:

$$\frac{dx_i}{dt} = H_i\left(x_i^*, \sum_{j=1}^{N}(A_{ij} + \mathbb{1}_{\{(i,j)=(m,n)\}}\Delta) G(x_i^*, x_j^*)\right), \tag{2.14}$$

$$x_i(0) = x_i^*, \text{ for } i = 1, 2, ..., N. \tag{2.15}$$

Then, at $t = 0$, for each node i:

$$\left.\frac{dx_i}{dt}\right|_{t=0} = H_i\left(x_i^*, \sum_{j=1}^{N}(A_{ij} + \mathbb{1}_{\{(i,j)=(m,n)\}}\Delta) G(x_i^*, x_j^*)\right) \tag{2.16}$$

$$\geq H_i\left(x_i^*, \sum_{j=1}^{N} A_{ij} G(x_i^*, x_j^*)\right) = 0, \tag{2.17}$$

which follows because H_i is increasing in its second input (Asm. 3) and because G is nonnegative (Lemma 1). We proceed to show that dx_i/dt remains nonnegative for all times for every i. Then, because we assume that activities cannot grow to infinity (Asm. 1), we find there must be a nonzero fixed point $\boldsymbol{x}^\dagger \geq \boldsymbol{x}^*$. To this end, define a partition of the N nodes into three sets for every time t_0:

$$P_{t_0} = \left\{ i = 1, 2, ..., N : \left.\frac{dx_i}{dt}\right|_{t=t_0} > 0 \right\}, \tag{2.18}$$

$$C_{t_0} = \left\{ i = 1, 2, ..., N : \left.\frac{dx_i}{dt}\right|_{t=t_0} = 0 \right\}, \tag{2.19}$$

$$N_{t_0} = \left\{ i = 1, 2, ..., N : \left.\frac{dx_i}{dt}\right|_{t=t_0} < 0 \right\}. \tag{2.20}$$

Our goal is to show that, starting with $N_0 = \emptyset$, we have $N_t = \emptyset$ for all $t > 0$. Note that if at any point have $P_t (= N_t) = \emptyset$, we are done: all x_i are constant in time.

Suppose, to the contrary, that at some $f > 0$, we have $N_f \neq \emptyset$. Let $s = \inf\{t : N_t \neq \emptyset\}$ be the first time that a node starts to decrease, noting that $0 \leq s \leq f$ so s is finite and nonnegative. Note that $N_s = \emptyset$, otherwise, if $j \in N_s$, we have by the intermediate value theorem that there is a $s' \in [0, s)$ such that,

$$\left.\frac{dx_j}{dt}\right|_{s'} = \frac{1}{2}\left.\frac{dx_j}{dt}\right|_{s} < 0, \tag{2.21}$$

because $dx_j/dt \geq 0$ at $t = 0$, which contradicts s's minimality. So, in fact, every node $i \in P_t \cup C_t$ for all $t \leq s$.

Let k be a node such that there is an $\varepsilon > 0$ such that $k \in N_t$ for every $t \in (s, s+\varepsilon)$, that is, we want k to be the first node that has negative time derivative, *nota bene* there may be more than one such node. We can find such a node by definition of s as an infimum. Taking the total time derivative of H_i, we find:

$$\frac{dH_i}{dt} = \frac{d}{dt}\left(\frac{dx_i}{dt}\right) = \sum_j \frac{\partial H_i}{\partial x_j}\frac{dx_j}{dt} = \frac{\partial H_i}{\partial x_i}\frac{dx_i}{dt} + \sum_{j \neq i} \frac{\partial H_i}{\partial x_j}\frac{dx_j}{dt}. \tag{2.22}$$

In particular, at $t = s$, we find:

$$\frac{d}{dt}\left(\frac{dx_i}{dt}\right) = \frac{\partial H_i}{\partial x_i}\frac{dx_i}{dt} + \sum_{j \in P_s \setminus \{i\}} \frac{\partial H_i}{\partial x_j}\frac{dx_j}{dt}, \tag{2.23}$$

where we only sum over nodes in P_s because those in C_s have zero time derivative so do not contribute. Now, there are two cases for our node at $t = s$: $k \in P_s$ and $k \in C_s$. We start with the easier case.

Suppose that $k \in P_s$. That is, $dx_k/dt > 0$ at s. But then, by continuity, there is an $\varepsilon' > 0$, such that $dx_k/dt > 0$ for every $t \in [s, s+\varepsilon')$, contradicting our assumption on k that immediately after $t = s$ it would have negative time derivative.

Suppose that $k \in C_s$. We find that,

$$\frac{d}{dt}\left(\frac{dx_k}{dt}\right) = \sum_{j \in P_s} \frac{\partial H_k}{\partial x_j}\frac{dx_j}{dt} \geq 0, \tag{2.24}$$

where in the inequality we used Lemma 2. If $d^2x_k/dt^2 > 0$, then by continuity there is an $\varepsilon' > 0$ such that dx_k/dt is strictly increasing on $(s - \varepsilon', s + \varepsilon')$, so that

$$\left.\frac{dx_k}{dt}\right|_{t = s - \varepsilon'/2} < \left.\frac{dx_k}{dt}\right|_{t=s} = 0 \tag{2.25}$$

but by definition of s, $dx_k/dt \geq 0$ for all $t \leq s$, a contradiction.

So for every $i \in C_s$, we see $d^2x_i/dt^2 = 0$ which implies $\partial H_i/\partial x_j = 0$ for every $j \in P_s$. However, inspection of the proof of Lemma 2 combined with Asm. 5 forces $A_{ij} = 0$ for each of these i, j. (Or P_s must be empty, but then we are done!) Because A is symmetric (Asm. 6), this means P_s and C_s are completely isolated from each other, it should be no surprise that the nodes in C_s are, in fact, already in a fixed state.

Because we assume each H_i is sufficiently smooth, it is at least locally Lipschitz continuous, thus there is an $h > 0$ such that there is a unique solution $x_i(t)$ to the ODE (Equation 1.12) for $t \in [s, s+h)$ [9]. However, we claim that for $t \in [s, s+h)$,

$$y_i(t) = \begin{cases} x_i(s) \text{ for } i \in C_s, \\ x_i(t) \text{ for } i \in P_s, \end{cases} \tag{2.26}$$

satisfies the differential equation. That is, we can keep x_i constant for $t > s$ for $i \in C_s$. Indeed, for $i \in P_s$,

$$H_i(y_i(t), \sum_j A_{ij} G(y_i(t), y_j(t))) = H_i(x_i(t), \sum_{j \in P_s} A_{ij} G(x_i(t), x_j(t))) = \frac{dy_i}{dt}, \tag{2.27}$$

where we used Asm. 6 to conclude $A_{ij} = 0$ for $j \in C_s$, and where the last equality holds by assumption that $x_i(t)$ is a solution. For $i \in C_s$ we find,

$$H_i(y_i(t), \sum_j A_{ij} G(y_i(t), y_j(t))) = H_i(x_i(s), \sum_{j \in C_s} A_{ij} G(x_i(s), x_j(s))) = 0 = \frac{dy_i}{dt}, \tag{2.28}$$

where the final equality holds by definition of $i \in C_s$. Because the solution is unique, this *is* the solution.

However, this contradicts our assumption that $dx_k/dt < 0$ immediately after time s. Thus at no point in time is N_t non-empty, and we are done.

■

2.6 Collapse: $\rho < \lambda^*$

We move on to proving the collapse bound: if $\rho < \lambda^*$, i.e. the symmetric network with $\lambda = \rho$ collapses, then so does the original network (ρ defined here: Def. 2). We prove the contrapositive: if the original network has a nonzero solution, then so does the $\lambda = \rho$ symmetric network (and so $\lambda^* \leq \rho$). Fortunately, we do not encounter any problems with the ambiguity of what a symmetric network is. However, we do first need two more lemmas from linear algebra, which we let the reader find in for example [6], and there is one additional assumption we have to make on the dynamics of the system.

Lemma 3. *For two matrices A, B such that $0 \leq B_{ij} \leq A_{ij}$, for all i, j, we have $\rho(B) \leq \rho(A)$.*

Lemma 4. *For A a nonnegative matrix and vector $\boldsymbol{x}$ there is an index j such that $(A\boldsymbol{x})_j \leq \rho(A) x_j$.*

Assumption 7. *The interaction term in our system equation (1.12) can be split into a product of two independent functions:* $G(x_i, x_j) = m(x_i)n(x_j)$.

Here $\rho(A)$ is the largest eigenvalue of A which is guaranteed to be real by the Perron-Frobenius theorem. The assumption on $G(x_i, x_j)$, while strong, holds in many systems, for example in the Cowan-Wilson model from 1.1.

Now, we can prove the collapse bound.

Proposition 5. *When* $\lambda^* > \rho$, *the original network has no nonzero fixed point.*

Proof. (Wu et al.) Suppose the original network has a solution $\boldsymbol{x}^*$. We want to find a nonzero solution for the symmetric network, so we regard the subset of vertices which have strictly nonzero equilibrium activity. Relabeling nodes if necessary, we find $x_1^*, x_2^*, ..., x_n^* > 0$ and $x_{n+1}^*, ..., x_N^* = 0$, for some $1 \leq n \leq N$. Let B be the induced subgraph on the first n nodes.

Now, because $\rho(B) = \rho\left(\begin{bmatrix} B & 0 \\ 0 & 0 \end{bmatrix}\right)$, Lemma 3 gives us that $\rho(B) \leq \rho(A)$. Furthermore, we see that $\boldsymbol{x}^*$ is also a stable solution for B because $G(x_i, 0) = 0$ (Asm. 4), so that

$$\sum_j A_{ij} G(x_i^*, x_j^*) = \sum_j B_{ij} G(x_i^*, x_j^*), \tag{2.29}$$

and thus $dx_i/dt = 0$ for all $i = 1, ..., n$. Finally, by Lemma 4 there is an index $k \in \{1, ..., n\}$ such that,

$$\sum_j B_{kj} n(x_j^*) = (Bn(\boldsymbol{x}^*))_k \leq \rho(B) n(x_k^*), \tag{2.30}$$

noting that this $x_k^* > 0$ by definition. Putting this together, we find, for this k,

$$\left.\frac{dx_k}{dt}\right|_{\boldsymbol{x}^*} = H\left(x_k^*, \sum_j A_{kj} G(x_i^*, x_j^*)\right), \tag{2.31}$$

$$= H\left(x_k^*, m(x_k^*) \sum_j B_{kj} n(x_j^*)\right), \tag{2.32}$$

$$\leq H\left(x_k^*, m(x_i^*) \rho(B) n(x_k^*)\right), \tag{2.33}$$

where we used that H is increasing in its second variable (Asm. 3). Notice that $dx_k/dt = 0$ by definition of our network being at equilibrium. But now, we find that at $x_k^* > 0$, the following function of a single variable is greater than zero:

$$0 \leq H(x_k^*, \rho(A) G(x_k^*, x_k^*)). \tag{2.34}$$

And because this function becomes negative in the limit to infinity (Asm. 1), the intermediate value theorem ensures there is an $x^\dagger \geq x_k^* > 0$ such that $H(x^\dagger, \rho(A) G(x^\dagger, x^\dagger) = 0$, i.e. the symmetric network with $\lambda = \rho$, in the sense of the one-node interpretation, has a fixed point, as desired. ■

3

Rigorous Bounds Application

We have discussed Wu et al.'s rigorous bounds on cooperative systems sufficiently. It is time to put it to the test. For this, we have selected the dynamics of the Cowan-Wilson model of neuron activity, which was introduced in Section 1.1, and will be studied in a more physiologically accurate context in the next chapter. Indeed, to satisfy the assumptions necessary before we can apply the bounds, we have to limit ourselves to cooperative networks, meaning populations of excitatory neurons only. Unfortunately, as we mentioned in the introduction, Cowan and Wilson point out that it is crucial for both inhibitory and excitatory neurons to be included for a model to have relevance to real-world neurons. We thus defer physical interpretations and conclusions to the next chapter.

3.1 Model and Simulation

For a nonnegative adjacency matrix A, steepness parameter τ and threshold μ, we consider a system described by:

$$\frac{dx_i}{dt} = -x_i + \frac{\mathcal{S}\left(\sum_{j=1}^{N} A_{ij}x_j\right) - \mathcal{S}(0)}{1-\mathcal{S}(0)}, \tag{3.1}$$

where $\mathcal{S}(x) = 1/(1+e^{\tau(\mu-x)})$ is a smoothed-out, continuous step function centered around μ, which we shift and scale so that the resulting function is zero when incoming activity is zero and asymptotically approaches one as incoming activity grows. One can verify that the equation satisfies the assumptions laid out in Section 2.1. Applying the process laid out by Wu et al., we desire the smallest λ such that

$$0 = -x + \frac{\mathcal{S}(\lambda x) - \mathcal{S}(0)}{1-\mathcal{S}(0)} \tag{3.2}$$

has a positive solution, which we call λ^*. We solve this function numerically, but we can give a decent approximation. Indeed, we look for the smallest λ such that the line $y = x$ and the quotient in (3.2) cross at a point to the right of the y-axis. Well, the quotient is a smoothed-out step function, which we expect to first touch the $y = x$ line at the 'hump' it forms before it levels off at $y = 1$. We define this hump precisely as the point at which the second derivative is minimal. Simple (but tedious) calculus yields $\lambda^* \approx \mu + \ln(2 + \sqrt{3})/\tau$, but after performing a numerical curve-fit, we find that $\lambda \approx \mu + 2/\tau$ is a much better approximation.

3.2 Method

Our first goal is to generate a graph of 'critical points', a point in parameter space which separates system survival and collapse. A numerically simulated system is considered near stability if $|d\boldsymbol{x}/dt|_2 < 10^{-4}$, in which case the simulation ends. We choose a cut-off point of 10^{-1} so that a system near stability with $|\boldsymbol{x}|_2 \leq 10^{-1}$ is considered collapsed and vice versa. This was chosen *a posteriori*, as usually the norm of $\boldsymbol{x}$ near stability was considerably larger or smaller than 10^{-1}.

We generate an Erdős-Renyi random graph with parameter p, after which we adjust its weights its multiplying them by an exponentially distributed factor, with rate λ. The initial value was uniformly randomized between 0.9 and 1.

Finally, for a range of values of τ, we find—and plot as a black line—the value of μ which corresponds to a critical point (by the bisection method). For the same τ, we calculate the μ's such that $\lambda^* = k_{\max}$, for the survival bound, and $\lambda^* = \rho$, for the collapse bound. We plot all three lines and shade the area where the bounds predict a network will survive in green and collapse in red.

3.3 Results

We have picked two of the resulting graphs to be displayed in Fig. 3.1. Both regard a system of $N = 30$ nodes, with the Erdős-Renyi parameter at $p = {}^{1}\!/_{29}$ and $p = {}^{26}\!/_{29}$, such that the expected degrees are 1 and 26 respectively for 3.1a and 3.1b. We set $\lambda = 1$ when we adjust weight strength. Right out the gate, it is clear both tipping-point lines respect the rigorous bounds. The lines start near the collapse bound for small τ, and asymptotically approach the survival bound with increasing τ. The dense network does this faster than the sparse one.

3.4 Discussion

A reader with sharp eyes may have noticed that it seems the black line actually *crosses* the red bound in the sparse network. This is simply a question of error tolerance, which, when calculating λ^* and the critical point, was set at 10^{-2}.

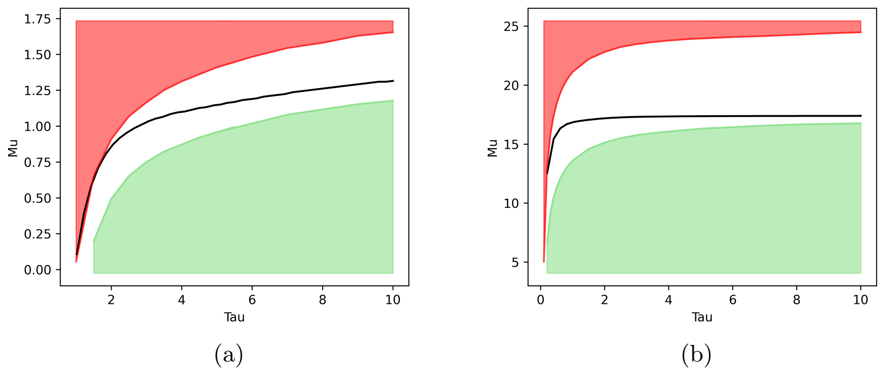


Figure 3.1: Two plots, each displaying the regions where the rigorous bounds predict a network survives and collapses, in green and red respectively, and a black line with the actual tipping point observed as described in Section 3.2, all in terms of the dynamics parameters τ and μ. For μ beneath the black line, the generated network had fixed points with norm $|\boldsymbol{x}| > 10^{-1}$, above, all solutions had norm below this cut-off. Both plots were generated on a system of $N = 30$ nodes, but (a) was retrieved from a network with expected degree 1, a sparse network, whereas (b) had expected degree 26, a dense one. Both tipping-point lines respect the bounds (up to numerical error).

Therefore, the apparent crossing is a minor discrepancy that falls within the acceptable margin of error and we do not consider it a failure of the collapse bound.

The more interesting question is why do both (and all observed) lines start touching the collapse-bound and approach the survival-bound. This is also discussed in Wu et al.'s paper. We start with the latter, which is a consequence of $\mathcal{S}$ approaching a step-function in the limit $\tau \to \infty$. Briefly: in this limit, one can use simple manipulations to prove that if the original network survives so does a $k_{\max}$-symmetric network, which combined with our survival bound implies that their tipping points coincide. The former is a consequence of whether the system undergoes a continuous transition at the point $\boldsymbol{x} = \mathbf{0}$, i.e. whether minimally solutions to Equation 1.13 form a continuous line as the parameter λ is continuously varied. Well, it turns out this is not the case for (3.2), which always has solution $x = 0$, but undergoes a so-called saddle-node bifurcation at $\lambda = \lambda^*$ where a positive solution appears, and splits into two points for $\lambda > \lambda^*$. We conclude that the conditions of the paper are sufficient but not necessary.

A final point is that it seems as though the denser network approaches the $k_{\max}$-bound faster, which is unexpected as the shape of $\mathcal{S}$ is independent of the size of the network, that is, the speed at which $\mathcal{S}$ approaches a step function depends exclusively on τ. We further investigate the matter by regarding the difference between each transition line to its respective $k_{\max}$ bound, displayed

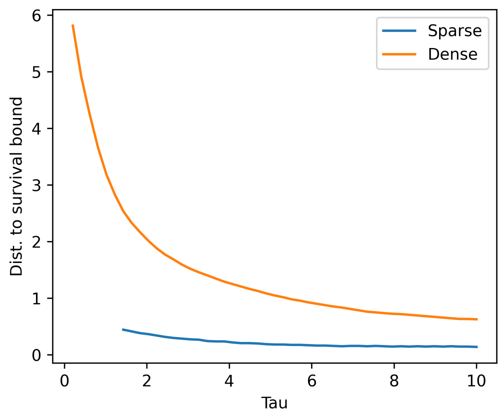


Figure 3.2: A plot showing the distance from the tippping-point line to the $k_{\max}$-bound line in Fig. 3.1, where 'sparse' refers to 3.1a and 'dense' to 3.1b. While the sparse net is in absolute terms closer to its bound, the dense net approaches faster. This can be explained by the shifting and scaling effect in Z that only becomes noticeable when the product $\mu\tau \gtrapprox 3$.

in Fig. 3.2, showing the rate of convergence. Although, in absolute terms, the sparse network is closer to its bound, over the same range of τ, the dense network's distance decreases faster. It seems as though, for the same τ, the sparse network converges slower. An explanation is the shifting and scaling which becomes significant as the product $\mu\tau$ becomes less than approximately three, which is the case in 3.1a for $\tau \lessapprox 3$. If this has the effect of decreasing the tipping point in terms of μ, then it explains why the sparse decrease in Fig. 3.2 is smaller: the starting value was lower. And indeed, simple algebraic manipulation[1] shows that the applied shifting and scaling never increases a sigmoid function, but it does decrease it significantly (for small x) when $\mu\tau \lessapprox 3$, so that each neuron receives less incoming activity, and so it collapses sooner, i.e. for smaller values of μ.

[1]We claim $\mathcal{S}(x) - \mathcal{S}(0) \leq \mathcal{S}(x)(1 - \mathcal{S}(0))$, which holds if $0 \leq \mathcal{S}(x) \leq 1$ as is the case.

4

Cowan-Wilson

In this chapter we take a more physically relevant look at the Cowan-Wilson model that was introduced in Section 1.1 and used as a benchmark for Wu et al.'s rigorous bounds in the previous chapter. First, we give a brief derivation of the original Cowan-Wilson model to place the equations we have used so far in a physiological context, explaining what assumptions underlie it and precisely what it describes. Next, we take inspiration from Laurence et al. and derive a one-dimensional reduction for a generalized version of the nonlinear, noncooperative model, by introducing several changes to their method. We test the accuracy of the reduction and how the error depends on several system parameters. Armed with this knowledge, we draw physical conclusions based on the reduced differential equation. Finally, we close the chapter by discussing our findings.

You can find examples of the model and reduction in Appendix B.

4.1 The Cowan-Wilson Model

In 1972, in a seminal paper, J.D. Cowan and H.R. Wilson define and study the so-called Cowan-Wilson (or Wilson-Cowan) model of the firing rate activity of a population of neurons [2]. Here, 'firing' refers to the generation of an action potential, where the voltage across the cell membrane of a neuron spikes (for a complete overview see [10]). The model consists of exactly two variables, $E(t)$ and $I(t)$, which represent the proportion of 'excitatory' and 'inhibitory' neurons respectively firing at a given time t. As the name suggests, excitatory neurons stimulate connected neurons to fire when they fire, while inhibitory neurons inhibit other neurons from firing. There is an underlying assumption here that is worth mentioning.

Neurons communicate with one another via synapses, where chemicals called neurotransmitters are exchanged. It depends on the type of neurotransmitter released (and the receptors they bond to) whether the receiving cell is excited or inhibited. A generally accepted rule of thumb (referred to as Dale's principle

[11]), which we assume to be true, is that a neuron releases the same type of neurotransmitter at all of its synaptic connections, meaning it is well-defined to classify and partition a population into excitatory and inhibitory neurons.

The equations the authors derive are:

$$E(t+h) = \left(1 - \int_{t-r_E}^{t} E(t')dt'\right) \mathcal{S}_E \left(\int_{-\infty}^{t} \alpha(t-t')(c_1E(t') - c_2I(t') + P(t'))dt'\right). \tag{4.1}$$

The equation for $I(t)$ can be obtained *mutatis mutandis*, so we leave it out for clarity. The equation is made up of a product of two distinct factors (and we comment on h later).

The left factor relates to the proportion of neurons that are ready to fire. This is a consequence of the 'refractory time' r_E, which is the duration after firing during which it is impossible to fire again, i.e. a neuron's 'cool-down period'. Thus, the integral $\int_{t-r_E}^{t} E(t')dt'$ gives the proportion of neurons that are in this cooldown period and unable to fire.

The right factor is comprised of an integral fed into an activation function, $\mathcal{S}_E$. The integral is a decay function α multiplied by a sum of incoming activity: c_1, c_2 are connection weights and $P(t')$ represents external stimuli, thus the integral represents incoming activity at time t.

We cover one interpretation of $\mathcal{S}$ given by the authors. (We dropped the subscript as the derivation holds for both variables.) We can heuristically model each *individual* neuron's activation function as a Heaviside function with variable threshold, θ. This gives rise to a threshold density function, $D(\theta)$. We can then define $\mathcal{S}$ as the distribution function over this density,

$$\mathcal{S}(x) = \int_{-\infty}^{x} D(\theta)d\theta, \tag{4.2}$$

where x is the total stimulus going into the population, thus we assume that all individual neurons receive the same amount of stimuli[1]. While $\mathcal{S}$ depends on the exact density $D(\theta)$, the authors choose

$$\mathcal{S}(x) = \frac{1}{1 + e^{\tau(\mu - x)}}, \tag{4.3}$$

for parameters τ and μ. This corresponds to a unimodal density centered around μ with width inversely related to τ, so that we can interpret μ as the average threshold and τ as a measure of spread of thresholds within a population.

A final note on the definition of E and I. The authors consider $E = I = 0$ as a state of low-level background activity, i.e. there are still neurons firing, as this appears universally in neural tissue. Thus small negative values of E and I still hold physiological relevance by reflecting a decrease in resting activity.

Now, to set up differential equations, the authors use a coarse grain approximation, where we replace any integrals by their integrand multiplied by a

[1]Conversely, the other interpretation the authors give is that all neurons have equal threshold but receive different amount of stimuli.

factor (essentially a moving average approximation), and we linearize $E(t+h) = E + hdE/dt$. This results in:

$$h\frac{dE}{dt} = -E + (1 - r_E E(t))\mathcal{S}(c_1 E(t) - c_2 I(t) + P), \tag{4.4}$$

where the authors additionally removed the time dependence of P. To ensure that zero is a fixed point, they shift $\mathcal{S}$ such that it is zero at zero, after which they rescale it so that it still asymptotically approaches one. We call this shifted and scaled function Z. The result starts to look like the one we used in Chapter 3.

Before we generalize this equation to N variables, we comment on further simplifications. First, we set $h = 1$, because this variable boils down to being a multiplicative factor of the derivative and thus only has the effect of determining the time-scale of a solution (regardless of the fact that we implicitly assumed h is small when we linearized E). Second, we set $r = 0$ for all nodes, meaning neurons have no cooldown period after firing. While not affecting the steps of the derivation, it is not immediately clear that this does not remove relevant behaviors from the system. Relying on numerical testing, we found that setting $r = 0$ did not remove limit cycles nor hysteresis. On the other hand, this step is crucial in deriving a reduction of dimension. Finally, in contrast to the Cowan-Wilson model, we make the assumption that every subpopulation has the same values for τ and μ. Similarly to setting $r = 0$, we cannot justify this assumption any more than by saying it anecdotally does not seem to remove behaviors. It is possible we are missing new behaviors, but for the sake of building a reduction, we continue with these assumptions now and address them in the Discussion later.

We are left with a basic model that we generalize to N subpopulations of neurons, by replacing $E(t)$ (or $I(t)$) by x_i, the c_i by A_{ij} and P (or Q, as the external stimulus of the inhibitory neurons is called) by p_i:

$$\frac{dx_i}{dt} = -x_i + Z\left(\sum_{j=1}^{N} A_{ij}x_j + p_i\right) := -x_i + \frac{\mathcal{S}\left(\sum_{j=1}^{N} A_{ij}x_j + p_i\right) - \mathcal{S}(0)}{1 - \mathcal{S}(0)}, \tag{4.5}$$

the difference here is that each x_i represents a separate subgroup; the population is not split into solely *all* excitatory and *all* inhibitory neurons. This way, one may hope to be able to model the interactions between several different parts of the brain at once. We can alternatively interpret this as dividing the excitatory and inhibitory populations in the original network into interacting subgroups; likely giving rise to more complex behaviors as a whole than observed in $E(t)$ (and $I(t)$).

In Fig. B.1, in the appendix, we highlight one feature of this model, its ability to generate periodic behavior without external stimulus. In the original model, the authors showed that P—the constant representing external stimulus into the group of excitatory neurons—has to be nonzero for oscillatory solutions. It turns out that solely 'interpopulation interactions' are sufficient in our model to cause oscillations, that is, every p_i set to zero.

4.2 One-dimensional Reduction

To derive the reduction, we follow a similar procedure to the one used in Laurence [1], with a few changes that come with interesting implications. For $\boldsymbol{a} \in \mathbb{C}^N$ s.t. $\sum_{i=1}^N a_i = 1$, define

$$R := \boldsymbol{a}^T \boldsymbol{x} = \sum_{i=1}^N a_i x_i. \tag{4.6}$$

Taking the time derivative and substituting Equation 4.5, we find

$$\frac{dR}{dt} = \sum_{i=1}^N a_i \frac{dx_i}{dt} = -R + \sum_{i=1}^N a_i Z\left(\sum_{j=1}^N A_{ij} x_j + p_i\right). \tag{4.7}$$

We aim to find conditions on $\boldsymbol{a}$ so that the dependency on x_j in Z vanishes. For this, we use the N-dimensional linear Taylor expansion of Z about the point $R\boldsymbol{b} + \boldsymbol{c}$ (for some $\boldsymbol{b}, \boldsymbol{c} \in \mathbb{C}^N$),

$$Z\left(\sum_{j=1}^N A_{ij} x_j + p_i\right) \approx Z\left(R\sum_{k=1}^N A_{ik} b_k + \sum_{k=1}^N A_{ik} c_k + p_i\right) + \\ \sum_{j=1}^N Z'\left(R\sum_{k=1}^N A_{ik} b_k + \sum_{k=1}^N A_{ik} c_k + p_i\right) A_{ij}(x_j - (Rb_j + c_j)), \tag{4.8}$$

which when we plug it back into (4.7), we find

$$\frac{dR}{dt} \approx -R + \sum_{i=1}^N a_i Z\left(R\sum_{k=1}^N A_{ik} b_k + \sum_{k=1}^N A_{ik} c_k + p_i\right) + \\ \sum_{i=1}^N \sum_{j=1}^N Z'\left(R\sum_{k=1}^N A_{ik} b_k + \sum_{k=1}^N A_{ik} c_k + p_i\right) a_i A_{ij}(x_j - (Rb_j + c_j)). \tag{4.9}$$

Now, if this were the linear case (Equation 1.1), the argument of Z and Z' would not be a function of i, and we could take them out of the summations. Fortunately, if we choose $\boldsymbol{b}$ and $\boldsymbol{c}$ such that $A\boldsymbol{b} = \alpha\mathbf{1}$ and $A\boldsymbol{c} = \gamma\mathbf{1} - \boldsymbol{p}$, for some α and γ, then we can remove the dependence on i (the input would equal $\alpha R + \gamma$). If A has an inverse, we can solve this exactly. In general, though, we will use linear least squares to determine $\boldsymbol{b}$ and $\boldsymbol{c}$, and assume this 'invertibility error' is small enough that it makes a good approximation.

Proceeding with the chosen $\boldsymbol{b}$ and $\boldsymbol{c}$, we find:

$$\frac{dR}{dt} \approx -R + Z(\alpha R + \gamma) + Z'(\alpha R + \gamma) \sum_{i=1}^N \sum_{j=1}^N a_i A_{ij}(x_j - (Rb_j + c_j)), \tag{4.10}$$

which results in the following conditions such that the linear term cancels:

$$\begin{aligned}\boldsymbol{a}^T A\boldsymbol{x} = \sum_{i=1}^N \sum_{j=1}^N a_i A_{ij} x_j &= \sum_{i=1}^N \sum_{j=1}^N a_i A_{ij}(Rb_j + c_j) \\ &= \boldsymbol{a}^T A(R\boldsymbol{b} + \boldsymbol{c}) \approx \alpha R + (\gamma - \boldsymbol{a}^T \boldsymbol{p}). \end{aligned} \tag{4.11}$$

Using the definition of R and leveraging that a scalar is invariant under the transpose, the above equation implies that $\boldsymbol{x}^T(A^T\boldsymbol{a}) = \boldsymbol{x}^T(\alpha\boldsymbol{a})$—if we take $\gamma = \boldsymbol{a}^T\boldsymbol{p}$. This is satisfied if $\boldsymbol{a}$ is an eigenvector of A^T with eigenvalue α, just as in the usual one-dimensional reduction.

For a suitable choice of $\boldsymbol{a}$ and α, we arrive at the following reduced system for R:

$$\frac{dR}{dt} \approx -R + Z(\alpha R + \boldsymbol{a}^T\boldsymbol{p}). \tag{4.12}$$

4.3 Selection of α

We have built the framework for a one-dimensional reduction. What remains is the choice of eigenvalue, α and normed eigenvector, $\boldsymbol{a}$. Here, the difference with the Laurence reduction is most evident, largely due to the model's nonlinearity. Recall in Equation 1.7 α multiplies the interaction term, motivating the choice of α to be as large as possible. In our reduction, the parameters affect only the argument of Z in (4.12), i.e. the expansion point of the linear approximation of Z. This approximation produces an error of order $\mathcal{O}[(x_j - Rb_j - c_j)^2]$ so that we should like $x_j - Rb_j - c_j \approx 0$ for every $j = 1, ..., N$. If we desire the total $\boldsymbol{a}$-weighted sum of such differences to equal zero, we find

$$R = \sum_{j=1}^N a_j x_j = R\sum_{j=1}^N a_j b_j + \sum_{j=1}^N a_j c_j, \tag{4.13}$$

meaning $\boldsymbol{a}^T\boldsymbol{b} = 1$ and $\boldsymbol{a}^T\boldsymbol{c} = 0$. Well as it turns out, both of these conditions are usually already satisfied.

We have assumed the 'invertibility error' defined above on $\boldsymbol{b}$ and $\boldsymbol{c}$ to be small. In fact it turns out random real matrices are invertible with probability one, meaning the error is indeed almost always negligible [12][2]. Now, we can expand each of $\boldsymbol{a}^T A\boldsymbol{b}$ and $\boldsymbol{a}^T A\boldsymbol{c}$, in two ways, using the definition of $\boldsymbol{a}$ and using the definition of $\boldsymbol{b}$ and $\boldsymbol{c}$:

$$\alpha\boldsymbol{a}^T\boldsymbol{c} \overset{\text{def}\,\boldsymbol{a}}{=} \boldsymbol{a}^T A\boldsymbol{c} \overset{\text{def}\,\boldsymbol{c}}{=} \gamma - \boldsymbol{a}^T\boldsymbol{p} = 0, \tag{4.14}$$

$$\alpha\boldsymbol{a}^T\boldsymbol{b} \overset{\text{def}\,\boldsymbol{a}}{=} \boldsymbol{a}^T A\boldsymbol{b} \overset{\text{def}\,\boldsymbol{b}}{=} \alpha. \tag{4.15}$$

[2]The intuition is that the determinant of a matrix is a polynomial function of its entries, and the Lebesgue measure of the zero set of a polynomial in the reals is zero, so that the determinant of a random matrix is zero with probability zero.

It follows that negligible invertibility error implies the conditions are satisfied. At the same time, if A is such that the invertibility error is large, it leads to linear terms not canceling out, creating an error of order $\mathcal{O}[x_j - Rb_j - c_j]$, which may make the model too inaccurate for use. Thus low invertibility error is crucial for the reduction.

So we are still left with the question which α to select. One may try to get smaller expansion point error by requiring b_j to be as close to 1 as possible, and choosing the α that accomplishes this. However, we conclude *a posteriori* that a similar idea, requiring the a_j to be close to $1/N$,

$$\alpha = \arg\min_{\lambda \in \sigma(A)} \{|\boldsymbol{v} - \mathbf{1}/N|_2 : A\boldsymbol{v} = \lambda v, \boldsymbol{v}^T \mathbf{1} = 1\}, \tag{4.16}$$

is the right way to go. We support and explain this in Section 4.5.1.

4.4 Simulation Parameters and Error Definition

In this section, we outline the process of generating Cowan-Wilson models to test our reduction and how we define the error of a reduction. An instance of the model is specified by four parameters: A, $\boldsymbol{p}$, μ, and τ. Below, we cover how we (randomly) generate each parameter.

We begin with the connection weight matrix A, where A_{ij} represents the coefficient of influence of node j activity on node i activity. To ensure the network satisfies Dale's principle (see Section 4.1), every node must be exclusively excitatory or inhibitory. Equivalently, every column of A must have all entries be positive or negative, respectively. We let p_{inhib} be the probability that a column is inhibitory. Additionally, in Cowan and Wilson's paper, it is generally large differences between the A_{ij} that lead to interesting behaviors, such as oscillatory solutions. To emulate this, we let their magnitudes be exponentially distributed: $|A_{ij}| \sim \text{Exp}(\lambda_A)$. Finally, we fix a probability p_0 of connections that are kept at zero, as we need not expect all nodes to interact with one another. The exact code can be found in e.g. Section C.

The $\boldsymbol{p}$ vector represents external stimuli. In the original two-dimensional model, the authors kept $p_2 = 0$ and only varied p_1. We thus postulate that it is again differences in stimuli that creates meaningful effects, and so we generate $p_i \sim \text{Exp}(\lambda_p)$, similarly to A_{ij} but with a larger rate, so that the p_i are generally smaller than the $|A_{ij}|$.

Lastly, to ensure all combinations of μ and τ are equally likely, we sample each from a uniform distribution.

Next, once an instance is defined, we choose an initial point $\boldsymbol{x}_0$ (usually randomly) and numerically solve (4.5) until $|d\boldsymbol{x}/dt|_2 < \varepsilon_{\text{stab}}$ where $\varepsilon_{\text{stab}} > 0$ is some small 'stability tolerance', so that an equilibrium point has been asymptotically reached, or we simulate until a fixed amount of time has passed (in case the solution is a limit cycle and never converges). To test the reduction, we select $\boldsymbol{a}$ and α and numerically solve (4.12) in the initial point $R_0 = \boldsymbol{a}^T \boldsymbol{x}_0$, stopping the simulation with the exact same conditions as $\boldsymbol{x}(t)$. The error is then defined as $|\boldsymbol{a}^T\boldsymbol{x}^* - R^*|$ (with x^* and R^* the final values of simulation).

Hyperparameters were chosen inspired by the values used in Cowan and Wilson's paper. In that paper there is one excitatory and one inhibitory subpopulation, so we set $p_{\text{inhib}} = 0.5$. We assume most subpopulations influence each other, so we set $p_0 = 0.2$, relatively low. The matrix entries in Cowan-Wilson are on the order of 10^0, so we set $\lambda_A = 1/3$ so that the average magnitude is 3. Lastly, again based on the values in Cowan-Wilson, we choose or randomize μ and τ between 0 and 5.

4.5 Accuracy of 1D Reduction

In this section we investigate the accuracy of the reduction, showing first the proposed method of selecting α is optimal, before computing error dependence on τ, μ and on the size of the network, to better understand any successes and shortfalls of the reduction.

4.5.1 Accuracy of α selection

We first show that the proposed method of selecting α is best. We compare five methods:

- Largest Eigenvalue (LE): select the eigenvalue with largest modulus.
- Smallest Eigenvalue (SE): select the eigenvalue with smallest modulus.
- Minimize $\boldsymbol{a}$ (MA): select α such that $|\boldsymbol{a}-\boldsymbol{1}/N|_2$ is minimized, as proposed in (4.16).
- Minimize $\boldsymbol{b}$ (MB): select α such that $|\boldsymbol{b}-\boldsymbol{1}|_2$ is minimized, to try to further reduce expansion point error.
- Random: select a random α and its corresponding eigenvector.

In each method $\boldsymbol{a}$ is chosen to be a corresponding eigenvector to α. We generate systems and define error as described in the previous section.

The results are displayed in Fig. 4.1a and Fig. 4.1b for $\boldsymbol{p}$ set to zero and $\boldsymbol{p}$ nonzero (generated with $\lambda_p = 10$) respectively. It is clear that minimizing $\boldsymbol{a}$ is the best performing method, both in terms of having the lowest mean absolute error, at around 0.2, and smallest uncertainty. The method performed significantly better than a random selection suggesting the importance of α selection. Lastly, we observe that for all methods there is no significant difference in error between the two plots.

We offer an explanation of MA's superiority. While R and all x_i are bounded above by one, $|\boldsymbol{a}|_2$ is can grow arbitrarily large while keeping $\boldsymbol{a}^T\boldsymbol{1} = 1$. Thus, $\boldsymbol{a}^T\boldsymbol{x}$ (and its difference with R) can grow arbitrarily large, pushing up both the

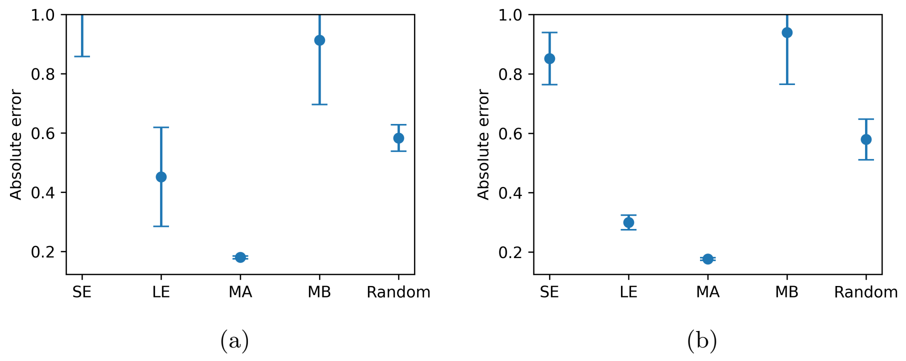


Figure 4.1: Two plots of mean absolute error (and uncertainty in the mean), between R and $\boldsymbol{a}^T\boldsymbol{x}$ in equilibrium or after a fixed amount of time, over 5,000 simulations per plot. Each simulation was carried out as explained in Section 4.4, where in (a) we set $\boldsymbol{p} = \boldsymbol{0}$ and in (b) we generated $\boldsymbol{p}$ with $\lambda_p = 10$. The optimal method was to find the weight vector $\boldsymbol{a}$ that most closely resembles the arithmetic average $\boldsymbol{1}/N$ (MA). Absolute errors can be arbitrarily large as R is bounded to $[0, 1]$ but $|\boldsymbol{a}|_2$ is unbounded.

average and the standard deviation of error (as error is nonnegative). Now, it turns out that because

$$|\boldsymbol{a} - \boldsymbol{1}/N|_2 = \sum_{i=1}^{N} a_i a_i^* - \frac{1}{N}\left(\sum_{i=1}^{N} a_i^* + a_i\right) + \frac{1}{N^2} = |\boldsymbol{a}|_2 - \frac{2}{N} + \frac{1}{N^2}, \tag{4.17}$$

when we apply the MA method, minimizing the left quantity, we are effectively minimizing the norm of $\boldsymbol{a}$, and thereby reducing the error caused by large $\boldsymbol{x}^T\boldsymbol{a}$. In fact, a look at the boxplot of errors (not pictured) of the data behind Fig. 4.1 shows that this method never produces the astronomical errors (upwards of 10^4) observed with other methods that let $|\boldsymbol{a}|$ become large.

A final point is on the interpretability of the MA method. By minimizing the distance to $\boldsymbol{1}/N$, we effectively find the most 'average' eigenvector, weighing every node as equally as possible. Thus, the corresponding weighted average R generally takes as many nodes into account as possible. In contrast, occasionally the LE method returns lower error than MA. However, inspection shows that in some of these cases $\boldsymbol{a}$ is heavily skewed towards some node m, so that $R \approx x_m$, i.e. the reduction is not relevant to the majority of nodes.

To conclude, we have shown the MA method produces the lowest average error and has the highest amount of interpretability. In particular, the error strongly depends on α selection. The rest of our simulations in this chapter continue with the MA method.

4.5.2 Unique Selection and Error

In some cases, the MA method gives more than one viable choice for α. Indeed, when α is complex, with eigenvector $\boldsymbol{a}$, we find that α^* with eigenvector $\boldsymbol{a}^*$ also satisfies MA, because $|\boldsymbol{a}^*|_2 = |\boldsymbol{a}|_2$ (combined with (4.17) from the previous section).

We ran 10,000 simulations with the MA method, similar as for Fig. 4.1, and kept track of error depending on whether α was unique or not. When α was unique, error was at 0.153 ± 0.003, compared to 0.251 ± 0.004 when it was not—almost twice as much. We interpret this to mean that systems with such a 'double α' are inherently unsuited for reduction to a single dimension. Also, we found that the method described in Section 4.4 generates unique and non-unique α networks with similar frequency (5,155 vs 4,845), suggesting that the reduction may inherently miss information for a staggering half of the generated networks. Thus, when α is complex, the reduction's accuracy is greatly reduced.

One explanation for this discrepancy is the fact that $\mathcal{S}(x)$ has a pole at $x = \mu \pm \tau^{-1}\pi i$, so that when R becomes complex near this point, the Taylor expansion loses its ability to approximate well. However, one would then expect that if τ is chosen small, say 10^{-1}, the poles would be distant enough from the usual domain of R that the reduction remains accurate. Surprisingly, we find the opposite is true: the reduction's error *increases* (regardless of μ) compared to $\tau = 1$ and remains significantly higher than when α is unique for the same τ.

We also attempted reducing error by employing a 'naive' two-dimensional model, where we calculate distinct one-dimensional reductions for each choice of α, and then average their activities. Because the choice of eigenvalues and eigenvectors are conjugate pairs, this has the effect of making R real. It had a positive effect, improving error slightly, but error remained significantly higher than when α was unique, likely because the separate reductions still suffer from inaccuracy due to being complex. The other option is a coupled two-dimensional reduction, where the two reduction variables depend on one another. We return to this in the Discussion.

4.5.3 Error-dependence on τ and μ

Next we investigate error dependence on τ and μ, shown in Fig. 4.2a. Network generation and simulation was done as described in Section 4.4, for $N = 6$ and $\boldsymbol{p} = \boldsymbol{0}$, but with some additions. Firstly, when a network results in non-unique α, we generate a new network. This greatly reduces the errors observed and allows us to study the parameter dependence of systems which we argue naturally call for a one-dimensional reduction.

The next addition is that once a system is generated, we run multiple simulations for different scaled copies of the network matrix A. That is, for each simulation we use $\kappa_n \cdot A$ where each κ_n is defined so that the real part of α of $\kappa \cdot A$ attains a certain value[3]. Then we calculate (approximately) the area

[3]These 'certain values' were determined using the approximation $\alpha^* = \mu + 2/\tau$ for α^* the smallest α such that (4.12) has a positive solution (which we infer to predict a tipping point).

between the two lines formed by $\boldsymbol{a}^T\boldsymbol{x}$ and R, as is depicted in Fig. 4.2b. The displayed error area in each point of Fig. 4.2a is the average taken over 50 such networks.

The resulting heatmap in Fig. 4.2a tells an interesting story. Once $\mu \gtrapprox 2$, the error is clearly an increasing function in μ and τ, steadily increasing and reaching its maximum in the top-right corner. The line $\mu\tau \approx 2$ breaks this pattern and it seems that for points with $\mu \lessapprox 1$, the error becomes independent of μ and τ. A minimum of error is seen for μ around 1.3 and τ greater than 2.

The second order term we drop out of the Taylor expansion (4.8) is a product of $Z''(\alpha R)$ and a sum depending on x_j among others. ($\gamma = 0$ because we take $\boldsymbol{p} = \boldsymbol{0}$.) Basic calculus shows the maximum absolute value of Z'' increases with τ^2, so that we expect the maximum second order error also increases with τ^2. However, the width of the interval where $Z''(x)$ is significantly nonzero decreases with τ; effectively when τ increases, we obtain two spikes, one for x just above μ and one just below, which, if crossed, can cause a large error which then propagates through the solution.

Besides depicting the area of error, Fig. 4.2b (and other such graphs) show that for unique α, our reduction is well able to predict system survival and collapse, in that the eigenvalue α captures enough information for this to be determined. See Fig. B.2 for contrasting plots when α is non-unique.

4.5.4 Error-dependence on N

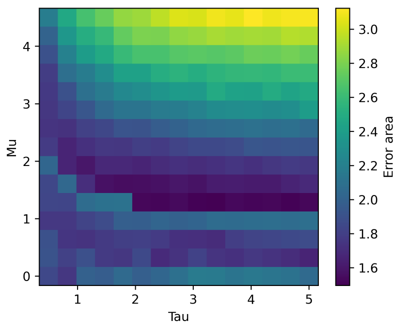
Finally, we take a look at error dependence as we change the size of the network. While it is natural to expect a dimension reduction to decrease in accuracy as we increase the scale of reduction, there may be an effect that counters this. We showed in Section 4.5.1 that our reduction relies heavily on selecting a particular eigenvector. When size increases, so does the number of eigenvector-eigenvalue pairs to choose from, which might lead one to conclude that the selection is more optimal, when compared to smaller N. We leave this to the results.

For each selected size, we generate 1,000 networks and randomize (τ, μ) uniformly in $[0.5, 5] \times [0, 5]$. Note, we do not filter networks based on their spectrum as in the previous section, as we aim to include as many potential sources of error as possible to assess the total dependence on size. The results can be found in Fig. 4.3, where, we see the average error grows steadily with network size. Another point is that the deviation in error remains largely constant, suggesting that the range of error is independent of the number of eigenvalues to choose from, i.e. our hypothesis seems not to hold.

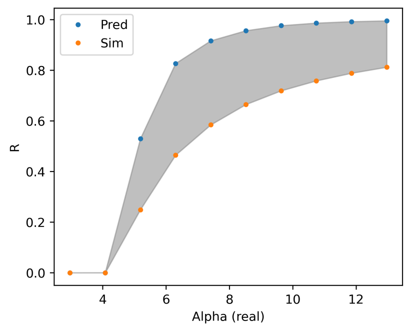
4.6 Analytical treatment of the reduction

So far, we have shown that, given a network, we can use the reduction to predict how the corresponding system will evolve. It would be valuable if we could do reverse this too: use the reduction to make predictions about systems with

Then we looked at $\text{Re}\{\alpha\} \in [\alpha^* - 2, \alpha^* + 8]$.



(a)



(b)

Figure 4.2: (a) Heatmap of the average absolute area of error between R and $\boldsymbol{a}^T\boldsymbol{x}$ for a system of $N = 6$ nodes, where $\boldsymbol{x}$ is the numerical solution of (4.5) as described in Section 4.4, with $\boldsymbol{p} = \mathbf{0}$, where we calculate error area over 10 simulations per network, for a range values of $\text{Re}\{\alpha\}$, as shown in (b). We restricted the program to networks that give a unique α as described in Section 4.5.2 and averaged over 50 such networks generated per μ, τ pair. (b) An example of a plot that depicts how the error area is defined, where 'sim' refers to $\boldsymbol{a}^T\boldsymbol{x}$ and 'pred' to R in equilibrium. Here $\tau = 0.5$ and $\mu \approx 1.0$ and the error area is roughly 2.4. Even though the prediction is off in magnitude, α seems to be a good indicator of whether the network survives (i.e. we have a well defined tipping-point around $\alpha \approx 5$). Note that α was unique in this plot, for non-unique plots see Fig. B.2.

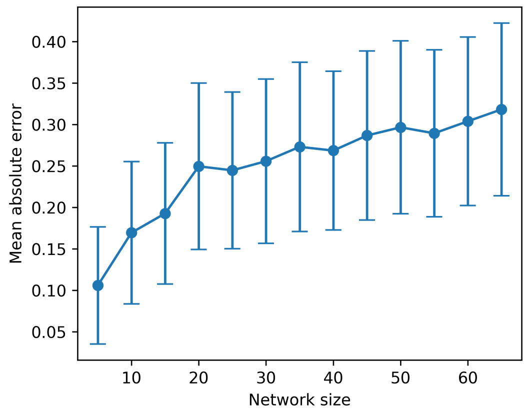


Figure 4.3: Plot of network size dependence of mean absolute error. Each dot represents the average error over 1,000 networks, generated as in Section 4.4. It is clear that error increases with size. The graph also shows how the standard deviation in error is largely independent of network size.

particular networks. We accomplish this by studying solutions of Equation 4.12 for parameter values where the reduction is accurate.

Having established that the reduction is most accurate for unique and thus real α and small μ and τ we will assume parameters are in this range. Because A is real, note that the corresponding eigenvector $\boldsymbol{a}$ is real and thus γ and R are too. In this case, we search for solutions of

$$0 = -R + \frac{\mathcal{S}(\alpha R + \gamma) - \mathcal{S}(0)}{1 - \mathcal{S}(0)}, \tag{4.18}$$

which is exactly Equation 3.2, with $\lambda \mapsto \alpha$ and $\mu \mapsto \mu - \gamma$ from the previous chapter. Using the approximation given in that chapter, we find that the minimal α for which the reduction has a nonzero solution is $\alpha^* \approx \mu - \gamma + 2/\tau$. In particular, this implies that the number of solutions is increasing in α. In comparison to Wu, we conclude that on the one hand, our reduction works for noncooperative networks and with $\boldsymbol{p} \neq \boldsymbol{0}$, but on the other, it clearly does not offer the same certainty as Wu's bounds.

When $R = u + iv$ is complex, the task changes significantly. Now, Equation 4.12 splits into two real differential equations. For one, we would hope that period solutions appear in this case (compared to the one-dimensional reduction). Unfortunately, we were unable to solve the equations analytically nor approximately, and plotting the isoclines $du/dt = 0, dv/dt = 0$ for various domains of u, v did not uncover any limit cycles.

4.7 Discussion

In this chapter we derived a generalized Cowan-Wilson model. To the best of the author's knowledge, this is a novel generalization, built on the same empirically-based assumptions as the original. For example, our model's behavior should be independent of the form of the chosen sigmoid function $\mathcal{S}$, which is critical to physiological application, as one cannot realistically determine the exact threshold density $D(\theta)$ besides an average and measure of spread (μ and τ). We say 'should' because this has not been explicitly tested, but it relies on the same premise used by Cowan and Wilson: changing the sigmoid function does not alter *behavior*, only *the circumstances* under which behaviors occur. Another key assumption was the application of Dale's principle when generating networks, as this was for the CW model *the* defining difference with other models of its time.

The resulting model is able to produce oscillatory behavior without external stimuli (in contrast to the original, where the authors showed $\boldsymbol{p}$ must be nonzero for such solutions to exist). We conclude it allows for the explanation of more complex behaviors, which may have experimental relevance, which gives a corresponding reduction value.

That said, the model was derived with the intention of extracting a reduction, so a few limiting assumptions were made. If the refractory time r were nonzero or if μ and τ were allowed to differ across subpopulations, the method of reduction would no longer work. We aimed for the model to produce oscillatory and hysteresis behavior, as in the original. While it succeeds at this, the model occasionally oscillates at extremely high frequencies, much higher than those reported in the original paper. Physically, the refractory time limits the minimum firing period of neurons, so the abnormally high frequencies are likely a consequence of ignoring r. We did not investigate whether differing τ and μ across groups causes significant differences, but the above argument for sigmoid independence suggests that having distinct τ and μ is equivalent to choosing different sigmoids.

This work can also be viewed as a case study of deriving and applying a Laurence reduction to a system of nonlinear ODEs outside of Laurence et al.'s framework. In fact, our method of reduction can immediately be generalized by replacing Z by an arbitrary function, as we do not rely on any qualities of Z (besides that it is sufficiently smooth). This is another confirmation that the exact form of $\mathcal{S}$ should not influence the model (and thus reduction), and it means this reduction may be applied to a wider range of models.

After extensive testing, we assess that, while the reduction is far from perfect, it is quite accurate *if* we consider only networks for which the MA method returns a unique α. The reduction is particularly useful for predicting whether a system will survive or collapse; when we directly analyze the reduction equation, we are given an approximate condition for the survival of a network, $\alpha \gtrapprox \mu + 2/\tau - \boldsymbol{a}^T\boldsymbol{p}$, relating the dynamics of the neuron population to network properties and external stimuli.

Indeed, the most pressing limitation of the reduction arises when α is not unique. It is yet unclear why the error increases disproportionately in these cases, but because the reduction may be applicable to a wide range of models, it is important to understand why. We suggest a more thorough investigation of the magnitude of the second derivative of Z when its argument is complex, as we only applied a heuristic approach. Furthermore, because a one dimensional, autonomous ODE has no periodic solutions, we effectively find that a unique reduction is not able to describe oscillatory behavior (however it still is able to describe the average activity of an oscillating system). Even when α was complex we did not find a reduction which oscillated (both by simulating solutions and by regarding isoclines). Thus the inherent issue may lie with the one dimensional reduction simply not capturing enough information, which calls for a multi-dimensional generalization, possibly similar to Laurence et al.'s approach: by choosing multiple R_k and approximating differential equations for each.

We conclude by listing out several interesting other topics which the author was not able to reach. It would be interesting to see them answered in further research, as each has major implications not only for the accuracy of our Cowan-Wilson model but, in light of the above, on a wider family of nonlinear dynamics.

We have not investigated error dependence on $\boldsymbol{p}$. This is important as the external stimulus played an essential part in the original model, so it is desirable for it to be well incorporated in the reduction.

Also, we only considered one family of networks to model, and we chose the hyperparameters that generated them relatively arbitrarily and did not study their impact on error. However, as these networks satisfied physiological constraints and produced a wide range of behaviors, it seems likely that this should not affect the results of the chapter.

It would be indicative to compare tipping-point line prediction accuracy. If the networks are cooperative, we can even directly compare this chapter's reduction to the Wu-bounds in the previous chapter.

Finally, it would be especially interesting to see if the interpretation at the bottom of Section 4.1 has any relevance. That is, if we define $E'(t) := \sum_{i \in \mathcal{E}} x_i / |\mathcal{E}|$, where $\mathcal{E}$ is the set of excitatory subpopulations, and similarly define $I'(t)$ and $\mathcal{I}$ for inhibitory subpopulations, we can ask whether $E'(t)$ and $I'(t)$ resemble the original Cowan-Wilson model variables, with each c_i chosen as the average or sum over internal or external connections between $\mathcal{E}$ and $\mathcal{I}$. We could go further and ask whether the original model can act *as a reduction* of the generalization. However, the feature that oscillations can occur even when $p = 0$ makes the selection of P and Q not obvious. Fitting into the Cowan-Wilson framework has the immediate advantage that we could relate a network to Cowan and Wilson's results and the half-a-century of results expanded on it.

In fact, one may be able to generalize this 'Cowan-Wilson' reduction to other noncooperative systems: by defining 'excitatory' and 'inhibitory' subpopulations as those with net positive and negative outgoing degrees, respectively, and deriving approximating differential equations for these. Of course, neural models applying Dale's principle seem especially suited for such a reduction, where the distinction between excitatory and inhibitory is clear, but one can imagine it may still work well in less clear systems.

5 Conclusion

Models based on ordinary differential equations, which describe complex systems of interacting components, unsurprisingly often produce complex and unpredictable behaviors. One solution is to reduce the dimension of the set of differential equations such that only a few remain. This has two main benefits: by retaining a few differential equations, it can still model specific behaviors; by reducing dimension we can numerically solve the system with significantly less computations and sometimes analytically derive conditions for solutions, both of which can be used to make predictions in the original system. We introduced and summarized the state-of-the-art reduction theory in three seminal papers, which we built on throughout this thesis. We also gave a review of the Cowan-Wilson model from computational neuroscience, building it from the ground up and highlighting key properties and assumptions made.

A key contribution of this thesis is the detailed and rigorous extension of Wu et al.'s proofs, where we provided necessary assumptions, lemmas, a discussion on the definition and solutions of a 'symmetric network,' and our proof of an important proposition. While a stronger version of this proposition is available through the theory on so-called 'quasimonotone' functions and their relation to differential inequalities, our focus remains on providing a proof that aligns with the scope of this thesis. The proof is by contradiction, and the author deems the final contradiction to be pleasing (although it requires two additional assumptions as mentioned).

We applied the freshly proven bounds to a special case of a generalization of the Cowan-Wilson model, focusing on strictly cooperative networks. The model is a generalization as it is defined on N-nodes, and the simulations were run specifically on strictly cooperative networks. We directly verified and confirmed all four claims Wu et al make about their bounds which we briefly discussed.

Finally, we derived our own one-dimensional reduction (inspired by Laurence et al) for the generalized Cowan-Wilson model, which is characterized by its nonlinear dynamics and necessarily noncooperative network. While we made several simplifying assumptions, the generalized model was able to produce the same results as the original, and more: while external stimulus was required for oscillatory solutions to exist in the original, in the generalized model where subpopulation interaction alone, under specific network and dynamic parameters, was sufficient. The one-dimensional reduction performs well, both in predicting activity and yielding approximate conditions for network collapse.

We believe Wu et al's proof has been rigorously established, furthermore their bounds functioned exactly as anticipated on the special case of the Cowan-Wilson model. Our final remarks focus on the implications and future of the Cowan-Wilson reduction.

It turns out that our method of reduction does not explicitly rely on the Cowan-Wilson function. Thus we have provided an extension of the Laurence reduction to a broader class of nonlinear dynamics. New research could look into whether our ideas can be advanced further to increase the variety of nonlinear dynamics reducible.

Although the reduction was one-dimensional, for around half of the generated networks the reduction parameter (that fully characterizes the reduction) was not unique, indicating these networks were inherently multi-dimensional. Indeed, in such cases, the average error was 66% larger than when the parameter was unique. These mean absolute errors were 0.25 compared to 0.15, highlighting at least that both are still relatively low, as error ranges between 0 and 1. This naturally calls for investigation into whether a multi-dimensional reduction exists in our framework and is able to improve the error discrepancy.

Our analysis of the Cowan-Wilson model essentially provides a case study of the limitations of a one-dimensional reduction for a nonlinear system with noncooperative networks—both qualities of which are critical to modeling real-world dynamics, but so far mostly ignored in the literature. Given the limitations observed, we expect that new reduction techniques might be necessary to obtain more robust predictions than here. We suggest as a starting point a 'Cowan-Wilson' reduction, splitting a network into 'excitatory' and 'inhibitory' nodes and then deriving approximating differential equations for each. Such a reduction may be able to better incorporate nonlinearity as well as noncooperativity, making it well worth further research.

Bibliography

[1] Edward Laurence et al. “Spectral Dimension Reduction of Complex Dynamical Networks”. In: *Phys. Rev. X* 9 (1 Mar. 2019), p. 011042. DOI: 10.1103/PhysRevX.9.011042. URL: https://link.aps.org/doi/10.1103/PhysRevX.9.011042.

[2] H.R. Wilson and J.D. Cowan. “Excitatory and Inhibitory Interactions in Localized Populations of Model Neurons”. In: *Biophysics Journal* 12 (1972). URL: https://www.cell.com/biophysj/pdf/S0006-3495(72)86068-5.pdf.

[3] Rui-Jie Wu et al. “Rigorous Criteria for the Collapse of Nonlinear Cooperative Networks”. In: *Phys. Rev. Lett.* 130 (9 Feb. 2023), p. 097401. DOI: 10.1103/PhysRevLett.130.097401. URL: https://link.aps.org/doi/10.1103/PhysRevLett.130.097401.

[4] Jianxi Gao, Baruch Barzel, and Albert-László Barabási. “Universal resilience patterns in complex networks”. In: *Nature* 530 (2016), pp. 307–312. URL: https://www.nature.com/articles/nature16948.

[5] Junjie Jiang et al. “Predicting tipping points in mutualistic networks through dimension reduction”. In: *Proceedings of the National Academy of Sciences of the United States of America* 115.4 (Jan. 2018), E639–E647. ISSN: 0027-8424. DOI: 10.1073/pnas.1714958115. URL: https://europepmc.org/articles/PMC5789925.

[6] R. A. Horn and C. R. Johnson. *Matrix Analysis*. Cambridge University, 1990.

[7] B. C. Daniels, S. T. M. Dissanayake, and B. R. Trees. “Synchronization of coupled rotators: Josephson junction ladders and the locally coupled Kuramoto model”. In: *Phys. Rev. E* 67 (2 Feb. 2003), p. 026216. DOI: 10.1103/PhysRevE.67.026216. URL: https://link.aps.org/doi/10.1103/PhysRevE.67.026216.

[8] G. Herzog and Lemmert R. *Differential inequalities*. 2001. URL: http://www.math.us.edu.pl/smdk/herzle.pdf.

[9] A. Coddington and N. Levinson. *Theory of Ordinary Differential Equations*. International series in pure and applied mathematics. McGraw-Hill, 1955. ISBN: 9780070992566. URL: https://books.google.be/books?id=bPJQAAAAMAAJ.

[10] E.R. Kandel et al. *Principles of Neural Science, Fifth Edition*. McGraw-Hill's AccessMedicine. McGraw-Hill Education, 2013. ISBN: 9780071390118. URL: https://books.google.be/books?id=s64z-LdAIsEC.

[11] V. J. Barranca et al. "Functional Implications of Dale's Law in Balanced Neuronal Network Dynamics and Decision Making". In: *Frontiers in neuroscience* (2022). DOI: https://doi.org/10.3389/fnins.2022.801847.

[12] Xinlong Feng and Zhinan Zhang. "The rank of a random matrix". In: *Applied Mathematics and Computation* 185.1 (2007), pp. 689–694. ISSN: 0096-3003. DOI: https://doi.org/10.1016/j.amc.2006.07.076. URL: https://www.sciencedirect.com/science/article/pii/S0096300306009040.

Appendix A

Noncooperative Networks

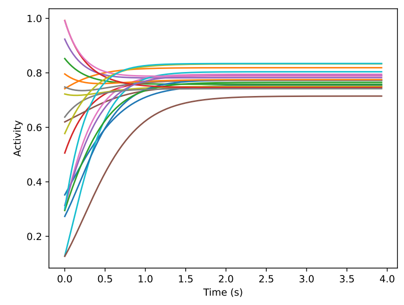
Allowing networks to have negative weights can cause drastic changes to its behavior. To illustrate such changes, we fix a network, initial value, and dynamic equation, and compare solutions of cooperative weights to noncooperative weights. These differences suggest the increase in difficulty of producing am accurate noncooperative reduction.

Consider N nodes governed by the equation,

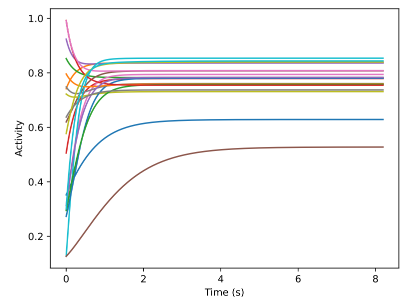
$$\frac{dx_i}{dt} = -x_i^3 + \sum_{j=1}^{N} A_{ij}(1-x_i)x_j, \tag{A.1}$$

which is a variant of the Susceptible-Infected-Susceptible model of the spread of disease used in Laurence [1]. Generate an Erdős-Renyi random graph, with parameter p, on the N nodes; thereby defining an adjacency matrix, $A_{ij} \in \{0,1\}$ for $i,j \in \{1,...,N\}$. Lastly, fix an initial vector with entries distributed uniformly between 0 and 1. Now, to generate the weights of the network, we replace $A_{ij} \mapsto \kappa_{ij}A_{ij}$, for each i,j, where κ_{ij} is an observation of $K \sim U(a,b)$. In other words, κ_{ij} is a (pseudo)random uniformly distributed number. To get a cooperative system, we choose any $0 < a \leq b$, for noncooperative systems, we let $a < 0 < b$. We will only allow these κ_{ij} to vary, all other parameters are held fixed.

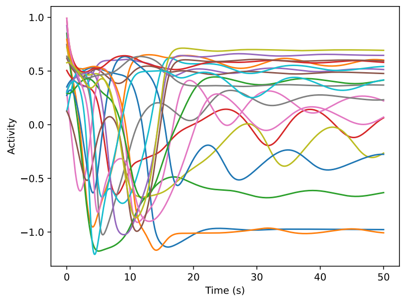
In Fig. A.1, we display the results. It is immediately clear that noncooperative networks allow more quickly changing and oscillatory behavior. Qualitatively, figures A.1a and A.1b are similar, even though their weights were largely different. The same cannot be said for figures A.1c and A.1d which have vastly different behaviors. Also notice that in both of the top plots, the network reached a fixed point in less than ten seconds, at which point the simulation was stopped. The bottom two reached the maximum of 50 seconds of simulation, but it seems both were heading towards an eventual fixed point.



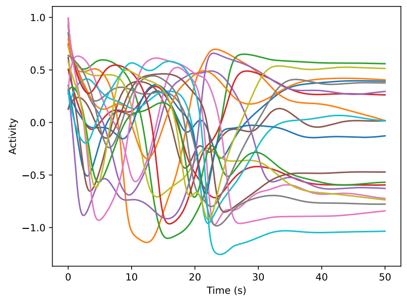
(a)



(b)



(c)



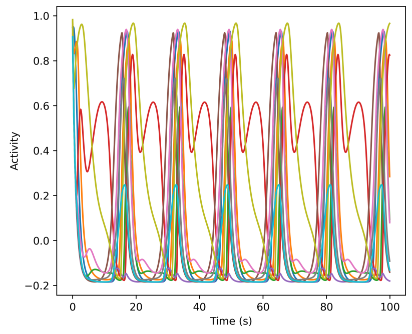
(d)

Figure A.1: Numerically simulated solutions of a Erdős-Renyi network (with $N = 20, p = {}^{6}/_{19}$) adhering to Equation A.1 with a fixed initial vector. Before simulating, the network weights were each perturbed by multiplication with a (pseudo)random uniformly distributed number. For the networks simulated in (a) and (b), the numbers were uniform between 0 and 1. For (c) and (d), these were between $-\frac{1}{2}$ and $\frac{1}{2}$. It is clear that the top and bottom graphs differ substantially to each other, and that the top two are comparable to each other, whereas the bottom two still differ. This represents the increase in complexity when allowing for non-cooperative networks.

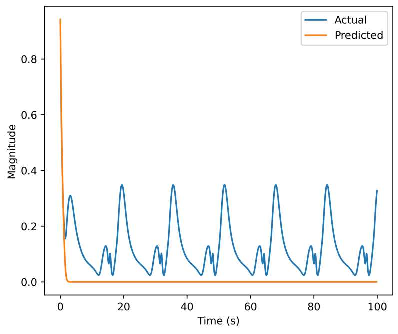
Appendix B

Examples of Cowan-Wilson Simulations and Reductions

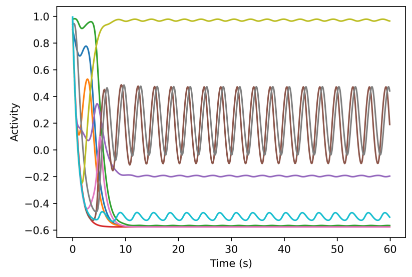
In this Appendix we display a few plots of the Cowan-Wilson model defined in Section 4.1 and the reduction we derived in Section 4.2. In Fig. B.1, we compare two oscillatory systems to their reductions. In Fig. B.2, we selected several graphs of error over a network parameter α.



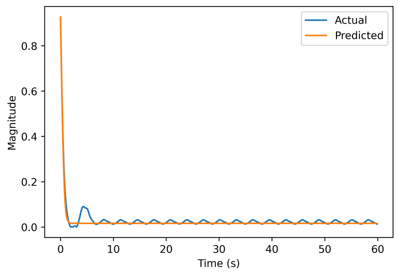
(a)



(b)

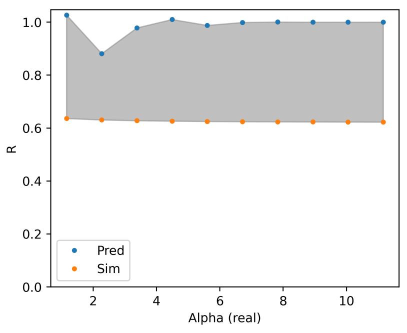


(c)

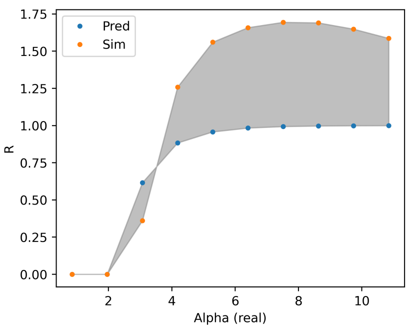


(d)

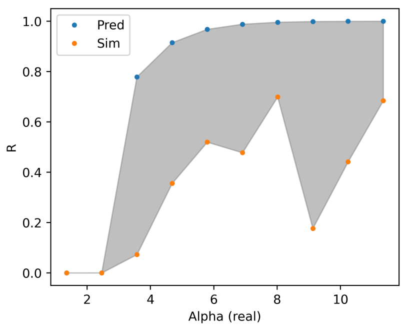
Figure B.1: Two oscillatory solutions of Equation 4.5 for $N = 10$ nodes without external stimulus ($\boldsymbol{p} = \boldsymbol{0}$) plotted in (a) and (c), along with the corresponding reduction in (b) and (d), where predicted R is plotted against $\boldsymbol{a}^T\boldsymbol{x}$ for $\boldsymbol{a}$ chosen as in Section 4.3. Each system (network and dynamic parameters) was generated as described in Section 4.4. In the original model, $P \neq 0$ was necessary for limit cycles to exist. We included the reduction to show that, although it does not oscillate, it still produces low error. There were also oscillating systems on which the reduction performed worse, but we did not investigate the average accuracy over these systems.



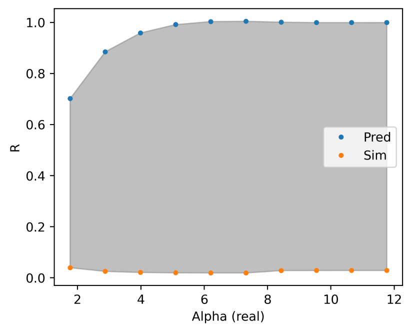
(a) Error area ≈ 4.02



(b) Error area ≈ 5.04



(c) Error area ≈ 8.41



(d) Error area ≈ 10.58

Figure B.2: Four plots of error area similar to Fig. 4.2b but for exclusively non-unique α and using the 'naive' two-dimensional reduction as described in Section 4.5.2 to calculate R_{pred} and R_{sim}. Note that R can be complex and the displayed area is between $|R_{\text{pred}}|$ and $|R_{\text{sim}}|$ whereas the error is calculated over $|R_{\text{pred}} - R_{\text{sim}}|$, which may cause the error to not 'fit' to the graph. The average error area with the employed parameters is a staggering 8.3 ± 0.3, compared to 2.5 ± 0.2 when we exclusively take unique α networks. Furthermore, we saw that non-unique α often provide no or little value as a predictor of R_{sim}, as in (c) and (d) where the two variables are especially uncorrelated, especially compared to plots with unique α.

Appendix C

Code

Fig. 4.1

```python
import json

import numpy as np
import time
from auxiliary_functions import exp, format_time_elapsed,
    plot_solution
from data_eater import reduce
from model import stability_run
import matplotlib.pyplot as plt

# Simulation parameters
DT = 1e-1
STABTOL = 1e-4
MAX_RUN_TIME = 100

# System parameters
SIZE = 10
PROB_INHIB = 0.5
P_0 = 0.2
CONNECTION_SCALE = 3
P_VECTOR_SCALE = 0
print(P_VECTOR_SCALE)

# Experiment parameters
NUM_POINTS = 5000


def Z(x, tau_, mu_):
    num = 1 / (1 + exp(tau_ * (mu_ - x))) - 1 / (1 + exp(tau_ * mu_
    ))
    den = 1 - 1 / (1 + exp(tau_ * mu_))
    return num / den


def largest_eig(A):
    """Take the largest one you can find"""
    eigs, vecs = np.linalg.eig(A.T)
    eig_i = np.argmax(np.abs(eigs))
    a, alpha = vecs[:, eig_i] / sum(vecs[:, eig_i]), eigs[eig_i]
    return a, alpha


def smallest_eig(A):
    eigs, vecs = np.linalg.eig(A.T)
    eig_i = np.argmin(np.abs(eigs))
    a, alpha = vecs[:, eig_i] / sum(vecs[:, eig_i]), eigs[eig_i]
    return a, alpha


def proposed_eig(A, flag_double_alpha=False):
    """The proposed solution to the eigenvalue selection problem"""
    eigs, vecs = np.linalg.eig(A.T)
    norms = []
    for ix in range(SIZE):
        a_ix = vecs[:, ix] / sum(vecs[:, ix])
        norms.append(np.linalg.norm(a_ix - 1))
    eig_i = np.argmin(norms)
    a, alpha = vecs[:, eig_i] / sum(vecs[:, eig_i]), eigs[eig_i]
    if flag_double_alpha:
        return a, alpha, (sorted(norms)[0] == sorted(norms)[1])
    return a, alpha


def minimize_b_eig(A):
    """Minimize the error in b=1"""
    eigs, vecs = np.linalg.eig(A.T)
    norms = []
    for ix in range(SIZE):
        b_ix = np.linalg.lstsq(A, eigs[ix] * np.ones(SIZE), rcond=
    None)[0]
        norms.append(np.linalg.norm(b_ix - 1))
    eig_i = np.argmin(norms)
    a, alpha = vecs[:, eig_i] / sum(vecs[:, eig_i]), eigs[eig_i]
    return a, alpha


def random_eig(A):
    """Randomly select an eigenvalue"""
    eigs, vecs = np.linalg.eig(A.T)
    eig_i = np.random.randint(SIZE)
    a, alpha = vecs[:, eig_i] / sum(vecs[:, eig_i]), eigs[eig_i]
    return a, alpha


METHODS = [smallest_eig, largest_eig, proposed_eig, minimize_b_eig,
     random_eig]


ERRORS = [[] for _ in range(len(METHODS))]
start, t1 = time.time(), time.time()
for q in range(NUM_POINTS):
    # Generate network
    double_alpha = True
    while double_alpha:
        columns = []
        for k in range(SIZE):
            col = np.random.choice([1., 0.], size=SIZE, p=[1 - P_0, P_0])
            col *= np.random.exponential(scale=CONNECTION_SCALE, size=SIZE)
            if np.random.random() < PROB_INHIB:
                col *= -1
            columns.append(col)
        A = np.column_stack(columns)
        double_alpha = proposed_eig(A, flag_double_alpha=True)[2]

    # Generate p vector
    P_VECTOR = np.random.exponential(scale=P_VECTOR_SCALE, size=SIZE)
    NONZERO_P_VECTOR: bool = (sum(p > 0 for p in P_VECTOR) > 0)

    # Compute invertibility error
    b_zero = np.linalg.lstsq(A, np.ones(SIZE), rcond=None)[0]
    inv_error = np.linalg.norm(A @ b_zero - np.ones(SIZE))
    if inv_error > 1e-2:
        print("Invertibility error on b too large:", inv_error)

    # W = (8 / 3 * np.random.random() + 1 / 3) * A  # Multiply A by a random number between 1/3 and 3
    W = A  # We remove this element of randomness to reduce deviation
    tau, mu = (np.random.random() * 4.5 + 0.5), (np.random.random() * 5)  # Random tau and mu
    for i, method in enumerate(METHODS):
        a, alpha = method(W)

        # Compute invertibility error for c
        if NONZERO_P_VECTOR:
            c = np.linalg.lstsq(W, (a @ P_VECTOR) * np.ones(SIZE) - P_VECTOR, rcond=None)[0]
            inv_error = np.linalg.norm(W @ c - (a @ P_VECTOR) * np.ones(SIZE) + P_VECTOR)
            if inv_error > 1e-2:
                print("Invertibility error on c too large:", inv_error)


        def model_f(arr):
            incoming_sum = W @ arr
            return -arr + [Z(incoming_sum[m], tau_=tau, mu_=mu) for m in range(len(arr))]


        def reduced_f(arr):
            return -arr[0] + Z(alpha * arr[0], tau_=tau, mu_=mu)


        x0s = [0.9 + 0.1 * np.random.random_sample(SIZE), 0.1 * np.random.random_sample(SIZE)]
        for x0 in x0s:
            sim = stability_run(model_f, DT, STABTOL, x0, max_run_time=MAX_RUN_TIME, debugging=0)
            pred = stability_run(reduced_f, DT, STABTOL, [a @ x0], max_run_time=MAX_RUN_TIME, debugging=0)

            ERRORS[i].append(np.abs(a @ sim[0] - pred[0][0]))

    if (q + 1) % int(NUM_POINTS / 100) == 0:
        print(
            f"At {(q + 1) // int(NUM_POINTS / 100)}%. Estimated {format_time_elapsed((time.time() - start) / (q + 1) * (NUM_POINTS - q - 1))} remaining")

NAME_DICT = {"largest_eig": "LE",
             "proposed_eig": "MA",
             "smallest_eig": "SE",
             "random_eig": "Random",
             "minimize_b_eig": "MB"}

# plt.boxplot(ERRORS, labels=[NAME_DICT[method.__name__] for method in METHODS])
# plt.ylabel("Error")
# plt.show()

with open("data/fig5.1error_list.txt","w") as f:
    json.dump(ERRORS, f)

plt.figure(figsize=(4, 3), dpi=600)
plt.errorbar([NAME_DICT[method.__name__] for method in METHODS], [np.mean(err) for err in ERRORS],
             [np.std(err) / np.sqrt(len(err)) for err in ERRORS], capsize=5, fmt='o', label='Mean')
plt.ylabel("Absolute error")
bottom, top = plt.gca().get_ylim()
if top > 1:
    plt.ylim(bottom, 1)
plt.savefig(f"data/fig5.1_pscale={P_VECTOR_SCALE}_numpoints={NUM_POINTS}.png")
plt.show()

# for i, error_list in enumerate(ERRORS):
#     plt.hist(np.array(error_list)/NUM_POINTS, label=NAME_DICT[METHODS[i].__name__], alpha=0.25)
# plt.legend()
# plt.xlabel("Error")
# plt.ylabel("Frequency")
# plt.show()

print("Phew, that took:", format_time_elapsed(time.time() - start))
```

Fig. 4.2

```python
import numpy as np
import time
from auxiliary_functions import exp, format_time_elapsed
from data_eater import reduce
from model import stability_run
import matplotlib.pyplot as plt

taus = np.linspace(0.5, 5, 15)
mus = np.linspace(5, 10, 15)
MUS, TAUS = np.meshgrid(mus, taus)
ERRORS = np.zeros(MUS.shape)

SIZE = 6
DT = 1e-1
STABTOL = 1e-4
MAX_RUN_TIME = 100
PROB_INHIB = 0.5
P_0 = 0.2
SCALE = 3
RUNS_PER_POINT = 50
NUM_ALPHAS = 10

print(f"{len(mus) * len(taus) * RUNS_PER_POINT * NUM_ALPHAS}
    simulations will be run at dt={DT}")

P_VECTOR = np.zeros(SIZE)  # Written here for clarity


def Z(x, tau_, mu_):
    num = 1 / (1 + exp(tau_ * (mu_ - x))) - 1 / (1 + exp(tau_ * mu_
    ))
    den = 1 - 1 / (1 + exp(tau_ * mu_))
    return num / den


def proposed_eig(A, speak=False):
    """The proposed solution to the eigenvalue selection problem"""
    eigs, vecs = np.linalg.eig(A.T)
    norms = []
    for ix in range(SIZE):
        a_ix = vecs[:, ix] / sum(vecs[:, ix])
        norms.append(np.linalg.norm(a_ix))
    eig_i = np.argmin(norms)
    a, alpha = vecs[:, eig_i] / sum(vecs[:, eig_i]), eigs[eig_i]
    if speak:
        return a, alpha, (sorted(norms)[0] == sorted(norms)[1])
    return a, alpha


start = time.time()
for q in range(RUNS_PER_POINT):
    # Generate network
    is_multi_dim = True
    while is_multi_dim:
        columns = []
        for k in range(SIZE):
            col = np.random.choice([1., 0.], size=SIZE, p=[1 - P_0,
     P_0])
            col *= np.random.exponential(scale=SCALE, size=SIZE)
            if np.random.random() < PROB_INHIB:
                col *= -1
            columns.append(col)
        A = np.column_stack(columns)
        a, alpha_zero, is_multi_dim = proposed_eig(A, speak=True)
    print(f"Eigenvalues run={q}: {np.linalg.eigvals(A)}")
    print(f"Chosen eig: {alpha_zero}, {a}")

    # Compute invertibility error (we assume P = 0)
    b_zero = np.linalg.lstsq(A, alpha_zero * np.ones(SIZE), rcond=None)[0]
    inv_error = np.linalg.norm(A @ b_zero - alpha_zero * np.ones(SIZE))
    if inv_error > 1e-2:
        print("Invertibility error too large:", inv_error)

    saved = 0
    # Run simulations
    for i in range(MUS.shape[0]):
        for j in range(MUS.shape[1]):
            mu = MUS[i, j]
            tau = TAUS[i, j]

            # Get predicted smallest alpha where system survives
            pred_alpha = mu + 2 / tau
            alphas = np.linspace(pred_alpha - 2, pred_alpha + 8, NUM_ALPHAS)
            dalpha = alphas[1] - alphas[0]

            # Run simulations and get total area of the difference between predicted and simulated R
            error_area = 0

            # # temp
            # preds, sims, diff = [], [], []
            for ideal_alpha in alphas:
                # Shift W to have the almost alpha
                W = ideal_alpha * A / np.real(alpha_zero)
                a, alpha = proposed_eig(W)


                def model_f(arr):
                    incoming_sum = W @ arr
                    return -arr + [Z(incoming_sum[m], tau_=tau, mu_=mu) for m in range(len(arr))]


                def reduced_f(arr):
                    return -arr[0] + Z(alpha * arr[0], tau_=tau, mu_=mu)


                x0 = 0.9 + 0.1 * np.random.random_sample(SIZE)
                sim = stability_run(model_f, DT, STABTOL, x0, max_run_time=MAX_RUN_TIME, debugging=0,
                                    title=f"simulation_tau={tau}_mu={mu}_run={q}_{round(time.time()) % 1000}")
                pred = stability_run(reduced_f, DT, STABTOL, [a @
    x0], max_run_time=MAX_RUN_TIME, debugging=0,
                                     title=f"reduction_tau={tau}_mu={mu}_run={q}_{round(time.time()) % 1000}")

                error_area += np.abs(a @ sim[0] - pred[0][0]) * dalpha

                # # temp
                # preds.append(np.abs(pred[0][0]))
                # sims.append(np.abs(a @ sim[0]))
                # diff.append(np.abs(a @ sim[0] - pred[0][0]))

            # # temp
            # if error_area > 2:
            #     saved += 1
            #     plt.figure(figsize=(5, 4), dpi=400)
            #     plt.plot(alphas, preds, '.', label='Pred')
            #     plt.plot(alphas, sims, '.', label='Sim')
            #     bottom, top = plt.gca().get_ylim()
            #     if 0 < bottom:
            #         plt.ylim(bottom=0)
            #     if top < 1:
            #         plt.ylim(top=1)
            #     plt.fill_between(alphas, preds, sims, alpha=0.5, color='grey')
            #     # plt.axvline(pred_alpha, color='r', label='Pred alpha')
            #     # plt.title(f"tau={tau}, mu={mu}")
            #     plt.xlabel("Alpha (real)")
            #     plt.ylabel("R")
            #     plt.legend()
            #     plt.savefig(f'data/fig5.2_tau={tau}_mu={mu}_error={round(error_area,1)}_double_choice={is_multi_dim}.png')
            #     plt.title(is_multi_dim)
            #     plt.show()

            # plt.figure(figsize=(5, 4), dpi=400)
            # plt.plot(alphas, diff, '.', label='diff')
            # plt.fill_between(alphas, [0 for _ in diff], diff, alpha=0.5, color='grey')
            # bottom, top = plt.gca().get_ylim()
            # if 0 < bottom:
            #     plt.ylim(bottom=0)
            # if top < 1:
            #     plt.ylim(top=1)
            # plt.xlabel("Alpha (real)")
            # plt.ylabel("R")
            # plt.legend()
            # plt.title(is_multi_dim)
            # plt.show()
            ERRORS[i, j] += error_area

    print(f"Estimated {format_time_elapsed((time.time() - start) / (q + 1) * (RUNS_PER_POINT - q - 1))} remaining\n")

ERRORS /= RUNS_PER_POINT

```

```python
# A heatmap of the error area for each tau and mu
plt.figure(figsize=(5, 4), dpi=600)
plt.pcolormesh(TAUS, MUS, ERRORS, shading='auto', cmap='viridis')
plt.colorbar(label='Error area')
plt.xlabel('Tau')
plt.ylabel('Mu')
plt.savefig(f'data/fig5.2_runs={RUNS_PER_POINT}_alphas={NUM_ALPHAS
    }.png')
plt.show()

print("Phew, that took:", format_time_elapsed(time.time() - start))
```

Fig. 4.3

```python
import json

import numpy as np
import time
from auxiliary_functions import exp, format_time_elapsed,
    plot_solution
from data_eater import reduce
from model import stability_run
import matplotlib.pyplot as plt

# Simulation parameters
DT = 1e-1
STABTOL = 1e-4
MAX_RUN_TIME = 100

# System parameters
PROB_INHIB = 0.5
P_0 = 0.2
CONNECTION_SCALE = 3
P_VECTOR_SCALE = 0

# Experiment parameters
SIZES = [_ for _ in range(5, 70, 5)]
NETWORKS_PER_SIZE = 1000
print(f"Running {len(SIZES)*NETWORKS_PER_SIZE}")


def Z(x, tau_, mu_):
    num = 1 / (1 + exp(tau_ * (mu_ - x))) - 1 / (1 + exp(tau_ * mu_
    ))
    den = 1 - 1 / (1 + exp(tau_ * mu_))
    return num / den


def proposed_eig(A, flag_double_alpha=False):
    """The proposed solution to the eigenvalue selection problem"""
    eigs, vecs = np.linalg.eig(A.T)
    norms = []
    for ix in range(A.shape[0]):
        a_ix = vecs[:, ix] / sum(vecs[:, ix])
        norms.append(np.linalg.norm(a_ix - 1))
    eig_i = np.argmin(norms)
    a, alpha = vecs[:, eig_i] / sum(vecs[:, eig_i]), eigs[eig_i]
    if flag_double_alpha:
        return a, alpha, (sorted(norms)[0] == sorted(norms)[1])
    return a, alpha


ERRORS_LIST = [[] for _ in range(len(SIZES))]
start = time.time()
for i, size in enumerate(SIZES):
    for q in range(NETWORKS_PER_SIZE):
        # Generate network
        # double_alpha = True
        # while double_alpha:
        columns = []
        for k in range(size):
            col = np.random.choice([1., 0.], size=size, p=[1 - P_0, P_0])
            col *= np.random.exponential(scale=CONNECTION_SCALE, size=size)
            if np.random.random() < PROB_INHIB:
                col *= -1
            columns.append(col)
        A = np.column_stack(columns)
        # double_alpha = proposed_eig(A, flag_double_alpha=True)[2]

        # Generate p vector
        P_VECTOR = np.random.exponential(scale=P_VECTOR_SCALE, size=size)
        NONZERO_P_VECTOR: bool = (sum(p > 0 for p in P_VECTOR) > 0)

        # Compute invertibility error
        b_zero = np.linalg.lstsq(A, np.ones(size), rcond=None)[0]
        inv_error = np.linalg.norm(A @ b_zero - np.ones(size))
        if inv_error > 1e-2:
            print("Invertibility error on b too large:", inv_error)

        tau, mu = (np.random.random() * 4.5 + 0.5), (np.random.random() * 5)  # Random tau and mu
        a, alpha = proposed_eig(A)

        # Compute invertibility error for c
        if NONZERO_P_VECTOR:
            c = np.linalg.lstsq(A, (a @ P_VECTOR) * np.ones(size) - P_VECTOR, rcond=None)[0]
            inv_error = np.linalg.norm(A @ c - (a @ P_VECTOR) * np.ones(size) + P_VECTOR)
            if inv_error > 1e-2:
                print("Invertibility error on c too large:", inv_error)


        def model_f(arr):
            incoming_sum = A @ arr
            return -arr + [Z(incoming_sum[m], tau_=tau, mu_=mu) for m in range(len(arr))]


        def reduced_f(arr):
            return -arr[0] + Z(alpha * arr[0], tau_=tau, mu_=mu)


        x0s = [0.9 + 0.1 * np.random.random_sample(size), 0.1 * np.random.random_sample(size)]
        for x0 in x0s:
            sim = stability_run(model_f, DT, STABTOL, x0, max_run_time=MAX_RUN_TIME, debugging=0)
            pred = stability_run(reduced_f, DT, STABTOL, [a @ x0], max_run_time=MAX_RUN_TIME, debugging=0)

            ERRORS_LIST[i].append(np.abs(a @ sim[0] - pred[0][0]))

    print(f"At {round(100 * (i + 1) / len(SIZES))}%. "
          f"Estimated {format_time_elapsed((time.time() - start) / (i + 1) * (len(SIZES) - i - 1))} remaining")

with open("data/fig5.3error_list.txt", "w") as f:
    json.dump(ERRORS_LIST, f)

plt.figure(figsize=(5, 4), dpi=600)
plt.errorbar(SIZES, [np.mean(error) for error in ERRORS_LIST],
             [np.std(ERRORS_LIST[i]) / np.sqrt(len(ERRORS_LIST)) for i in range(len(SIZES))], capsize=5, marker='o',
             label='Mean')
plt.ylabel("Mean absolute error")
plt.xlabel("Network size")
bottom, top = plt.gca().get_ylim()
if top > 1:
    plt.ylim(bottom, 1)
plt.savefig(f"data/fig5.3_pscale={P_VECTOR_SCALE}_numpoints={len(SIZES) * NETWORKS_PER_SIZE}.png")
plt.show()

print("Phew, that took:", format_time_elapsed(time.time() - start))
```