A mean-field toolbox for spiking neuronal network model analysis

Moritz Layer [1,2,*], **Johanna Senk** [1], **Simon Essink** [1,2], **Alexander van Meegen** [1,3], **Hannah Bos** [1], **Moritz Helias** [1,4]

[1] *Institute of Neuroscience and Medicine (INM-6) and Institute for Advanced Simulation (IAS-6) and JARA Institute Brain Structure-Function Relationships (INM-10), Jülich Research Centre, Jülich, Germany*
[2] *RWTH Aachen University, Aachen, Germany*
[3] *Institute of Zoology, Faculty of Mathematics and Natural Sciences, University of Cologne, Cologne, Germany*
[4] *Department of Physics, Faculty 1, RWTH Aachen University, Aachen, Germany*

Correspondence*:
Moritz Layer
m.layer@fz-juelich.de

ABSTRACT

Mean-field theory of spiking neuronal networks has led to numerous advances in our analytical and intuitive understanding of the dynamics of neuronal network models during the past decades. But, the elaborate nature of many of the developed methods, as well as the difficulty of implementing them, may limit the wider neuroscientific community from taking maximal advantage of these tools. In order to make them more accessible, we implemented an extensible, easy-to-use open-source Python toolbox that collects a variety of mean-field methods for the widely used leaky integrate-and-fire neuron model. The Neuronal Network Mean-field Toolbox (NNMT) in its current state allows for estimating properties of large neuronal networks, such as firing rates, power spectra, and dynamical stability in mean-field and linear response approximation, without running simulations on high performance systems. In this article we describe how the toolbox is implemented, show how it is used to calculate neuronal network properties, and discuss different use-cases, such as extraction of network mechanisms, parameter space exploration, or hybrid modeling approaches. Although the initial version of the toolbox focuses on methods that are close to our own past and present research, its structure is designed to be open and extensible. It aims to provide a platform for collecting analytical methods for neuronal network model analysis and we discuss how interested scientists can share their own methods via this platform.

Keywords: mean-field theory, (spiking) neuronal network, integrate-and-fire neuron, open-source software, parameter space exploration, (hybrid) modeling, Python, computational neuroscience, scientific computing

1 INTRODUCTION

Biological neuronal networks are composed of large numbers of recurrently connected neurons, with a single cortical neuron typically receiving synaptic inputs from thousands of other neurons (Braitenberg and Schüz, 1998; DeFelipe et al., 2002). Although the inputs of distinct neurons can be integrated in a complex fashion, average properties of entire populations of neurons often do not depend strongly on the contributions of individual neurons. Based on this observation, it is possible to develop analytically tractable theories of population properties, in which the effects of individual neurons are averaged out and the complex, recurrent input to individual neurons is replaced by a simplification, known as self-consistent effective input (Gerstner et al., 2014). In classical physics terms (e.g., Goldenfeld, 1992), this effective input is called *mean-field*, because it is the self-consistent mean of a *field*, which here is just another name for the input the neuron is receiving. The term *self-consistent* refers to the fact that the population of neurons that receives the effective input, is the same that contributes to this very input in a recurrent fashion: the population's output determines its input and vice-versa. The stationary statistics of the effective input therefore can only be found in a self-consistent manner: the input to a neuron must be set exactly such that the caused output precisely leads to the respective input.

These mean-field theories have been developed for many different kinds of synapse, neuron, and network models. They have been successfully applied to study average population firing rates (van Vreeswijk and Sompolinsky, 1996; van Vreeswijk and Sompolinsky, 1998; Amit and Brunel, 1997b), and the various activity states a network of spiking neurons can exhibit, depending on the network parameters (Amit and Brunel, 1997a; Brunel, 2000; Ostojic, 2014), as well as the effects that different kinds of synapses have on firing rates (Fourcaud and Brunel, 2002; Lindner, 2004; Schuecker et al., 2015; Schwalger et al., 2015; Mattia et al., 2019). They have been used to investigate how neuronal networks respond to external inputs (Lindner and Schimansky-Geier, 2001; Lindner and Longtin, 2005), and they explain why neuronal networks can track external input on much faster time scales than a single neuron could (van Vreeswijk and Sompolinsky, 1996; van Vreeswijk and Sompolinsky, 1998). They allow studying pair-wise correlations of neuronal activity (Sejnowski, 1976; Ginzburg and Sompolinsky, 1994; Lindner et al., 2005; Trousdale et al., 2012) and were able to reveal why pairs of neurons in random networks, despite receiving a high proportion of common input, can show low output correlations (Hertz, 2010; Renart et al., 2010; Tetzlaff et al., 2012; Helias et al., 2014), which for example has important implication for information processing. They describe pair-wise correlations in network with spatial organization (Rosenbaum and Doiron, 2014; Rosenbaum et al., 2017; Dahmen et al., 2021) and can be generalized to correlations of higher orders (Buice and Chow, 2013). Mean-field theories were utilized to show that neuronal networks can exhibit chaotic dynamics (Sompolinsky et al., 1988; van Vreeswijk and Sompolinsky, 1996; van Vreeswijk and Sompolinsky, 1998), in which two slightly different initial states can lead to totally different network responses, which has been linked to the network's memory capacity (Toyoizumi and Abbott, 2011; Schuecker et al., 2018). Most of the results mentioned above have been derived for networks of either rate, binary, or spiking neurons of a linear integrate-and-fire type. But various other models have been investigated with similar tools as well; for example, just to mention a few, Hawkes processes, non-linear integrate-and-fire neurons (Brunel and Latham, 2003; Fourcaud-Trocmé et al., 2003; Richardson, 2007, 2008; Grabska-Barwinska and Latham, 2014; Montbrió et al., 2015), or Kuramoto-type models (Stiller and Radons, 1998; van Meegen and Lindner, 2018). Additionally, there is an ongoing effort showing that many of the results derived for distinct models are indeed equivalent and that those models can be mapped to each other under certain circumstances (Grytskyy et al., 2013; Ostojic and Brunel, 2011; Senk et al., 2020).

Other theories for describing mean population rates in networks with spatially organized connectivity, based on taking a continuum limit, have been developed. These more phenomenological theories, known as neural field theories, have deepened our understanding of spatially and temporally structured activity patterns emerging in cortical networks. Starting with the seminal work by Wilson and Cowan (1972, 1973) investigating global activity patterns and Amari (1975, 1977) investigating stable localized neuronal activity. They were successfully applied to explain hallucination patterns (Ermentrout and Cowan, 1979; Bressloff et al., 2001), as well as EEG and MEG rhythms (Nunez, 1974; Jirsa and Haken, 1996, 1997). The neural field approach has been used to model working memory (Laing et al., 2002; Laing and Troy, 2003), motion perception (Giese, 2012), cognition (Schöner, 2008), and more; for extensive reviews of the literature, we refer the reader to Coombes (2005), Bressloff (2012), and Coombes et al. (2014).

Apparently, analytical theories have contributed to our understanding of neuronal networks and they provide a plethora of powerful and efficient methods for network model analysis. However, following the details behind those theories often requires some mathematical background. Furthermore, the analytical methods developed in those works only are of immediate use to other researchers if there is a numerical implementation that is flexible enough to be applicable to different network models, according to that researchers' questions. Such an implementation is often far from straightforward and at times requires investing substantial time and effort. Commonly, such tools are implemented as the need arises, and their reuse is often not organized systematically. Also reuse is often restricted to within a single lab. This way, not only are effort and costs spent by the neuroscientific community duplicated over and over again, but also are many scientists deterred from taking maximal advantage of those methods; simply because they do not have the capacities to do so, although such tools might open new avenues for investigating their research questions.

In order to make analytical tools for neuronal network model analysis accessible to a wider part of the neuroscientific community, and to create a platform for collecting well-tested and validated implementations of such tools, we have developed the Python toolbox NNMT (Layer et al., 2021), short for Neuronal Network Mean-field Toolbox. NNMT has been designed to fit the diversity of mean-field theories, and the key features we are aiming for are modularity, extensibility, and a simple usability. Furthermore, it features an extensive test suite to ensure the validity of the implementations as well as a comprehensive user documentation. The current version of NNMT mainly comprises tools for investigating networks of leaky integrate-and-fire neurons as well as some methods for studying neural field models. The toolbox is open-source and publicly available on GitHub[1].

There exist different approaches for neuronal network model analysis that do not use direct simulations of the subthreshold dynamics of individual neurons. Cain et al. (2016), for example, analyze the Potjans and Diesmann cortical microcircuit model (Potjans and Diesmann, 2014) employing the simulation platform DiPDE[2]. DiPDE is based on the population density approach and yields an efficient numerical method for analyzing the statistical evolution of voltage distributions of large homogeneous populations of neurons. Schwalger et al. (2017) start from a stochastic microscopic neuron model and derive mesoscopic population equations, which reproduce the statistical and qualitative behavior of the homogeneous neuronal subpopulations. But to the best of our knowledge, there currently is no toolbox pursuing efforts to collect numerical implementations of various analytical mean-field tools for neuronal network model analysis.

[1] https://github.com/INM-6/nnmt

[2] http://alleninstitute.github.io/dipde

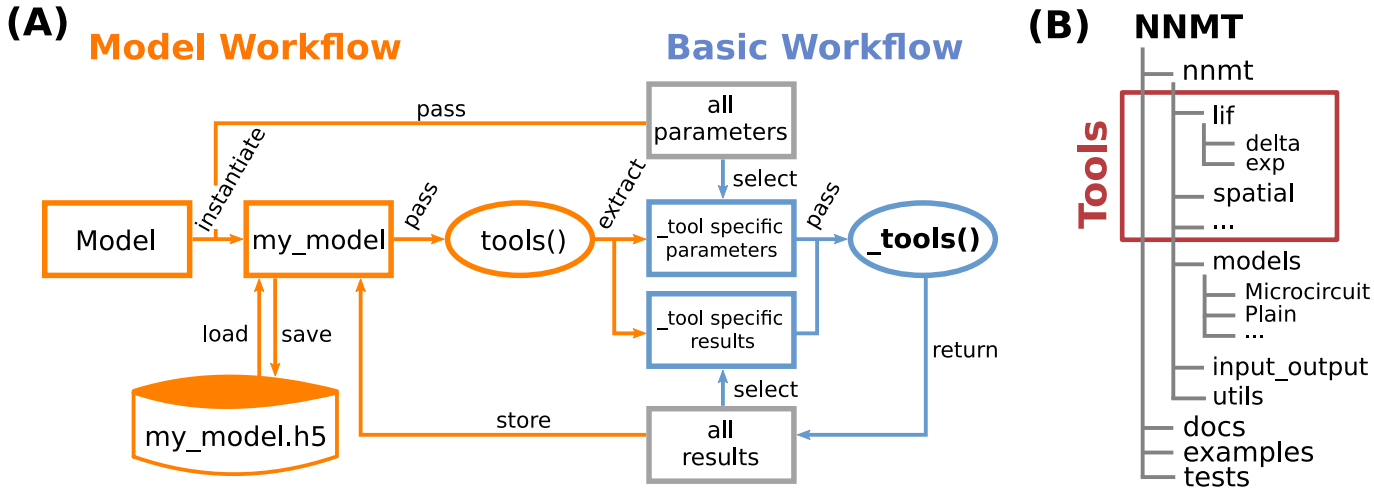

Figure 1. The Neuronal Network Mean-field Toolbox (NNMT) is a platform for collecting network analysis tools. **(A)** Individual tool functions can be used directly by explicitly passing arguments (basic workflow, blue) or via a convenience layer with a model class and wrapper functions facilitating the handling of parameters and results (model workflow, orange). **(B)** Structure of the repository. In addition to the tool collection (red frame, containing the `tools()` and the `_tools()`), pre-defined model classes, and utility functions in the NNMT Python package, the repository also comprises documentation, usage examples, and a test suite.

```
# basic workflow
result = nnmt.<submodule>.<_tool>(*args, **kwargs)

# model workflow
my_model = nnmt.models.<model>(<network_params>, <analysis_params>)
result = nnmt.<submodule>.<tool>(my_model)
```

Listing 1: The two modes of using NNMT: In the basic workflow (top), quantities are calculated by passing all required arguments directly to the underscored tool functions available in the submodules of NNMT. In the model workflow (bottom), a model class is instantiated with parameter sets and the model instance is passed to the non-underscored tool functions which automatically extract the relevant parameters.

In the following, we present the design considerations that led to the structure and implementation of NNMT as well as a representative set of use cases. Section 2 first introduces its architecture. Section 3 then explains its usage by reproducing previously published network model analyses from Sanzeni et al. (2020), Bos et al. (2016), Schuecker et al. (2015), and Senk et al. (2020). Section 4 discusses the role of community-wide accessible toolboxes for scientific software development in general and indicates current limitations and prospective advancements of NNMT.

2 WORKFLOWS AND ARCHITECTURE

What are the requirements a package for collecting analytical methods for neuronal network model analysis needs to fulfill? To begin with, it must be adaptable and modular enough to accommodate many and diverse analytical methods while avoiding code repetition and a complex interdependency of package components. It must enable the application of the collected algorithms to various network models in a simple and transparent manner. It needs to make the tools easy to use for new users, while also providing experts with direct access to all parameters and options. Finally, the methods need to be thoroughly tested and well documented.

These are the main considerations that guided the development of NNMT. **Figure 1A** illustrates that the toolbox can be used in to two different workflows depending on the preferences and goals of the user. In the *basic workflow* the individual method implementations called *tools* are directly accessed, whereas the *model workflow* provides additional functionality for the handling of parameters and results.

2.1 Basic Workflow

The core of NNMT is a collection of low-level functions that take specific parameters (or pre-computed results) as input arguments and return analytical results of network properties. In **Figure 1A**, we refer to such basic functions as `_tools()`, beginning with an underscore. We term this lightweight approach of directly using these functions the basic workflow. The top part of **Listing 1** demonstrates this usage; for example, the quantity to be computed could be the mean firing rate of a neuronal population and the arguments could be parameters which define neuron model and external drive. While the basic workflow gives full flexibility and direct access to every parameter of the calculation, it remains the user's responsibility to insert the arguments correctly, e.g., in the right units.

2.2 Model Workflow

The model workflow adds a convenience layer on top of the basic workflow (**Figure 1A**). A *model* in this context is an object that stores a larger set of parameters and can be passed directly to a `tool()`, the non-underscored wrapper of the respective `_tool()`. The `tool()` automatically extracts the relevant parameters from the model, passes them as arguments to the corresponding core function `_tool()`, returns the results, and stores them in the model. The bottom part of **Listing 1** shows how a model is initialized with parameters and then passed to a `tool()` function.

Models are implemented as Python classes and can be found in the submodule `nnmt.models`. We provide the class `nnmt.models.Network` as a parent class and a few example child classes which inherit the generic methods and properties but are tailored to specific network models; custom models can be added straightforwardly. The parameters distinguish network parameters, which define neuron models and network connectivity, and analysis parameters; an example for an analysis parameter is a frequency range over which a function is evaluated. Upon model instantiation, parameter sets defining values and corresponding units are passed as Python dictionaries or `yaml` files. The model constructor takes care of reading in these parameters, deriving dependent parameters, and converting all units to SI units for internal computations. Consequently, the parameters passed as arguments and the functions for deriving dependent parameters of a specific child class need to be aligned. This design encourages a clear separation between a concise set of base parameters and functionality that transforms these parameters to the generic (vectorized) format that the tools work with. To illustrate this, consider the weight matrix of a network of excitatory and inhibitory neuron populations in which all excitatory connections have the same weight and all inhibitory ones another weight. As argument one could pass just a tuple of these two different weight values and the corresponding model class would take care of constructing the full weight matrix.

When a `tool()` is called, the function will check whether all required parameters are available in the passed model object. If not, an error is raised, pointing the user towards the missing parameters. If the property to be computed is based on the computation of another property that has yet to be calculated, the `tool()` will also raise an error, informing the user of which other properties must be calculated first. This way, the user is guided towards the desired result. When all required parameters and results are available, the `tool()` extracts the required arguments, calls the respective `_tool()`, caches the result and its units into the network model's result dictionaries, and returns the result. If the user attempts to compute the same property twice, using identical parameters, the `tool()` will retrieve the already computed result from the cache and return that value. Results can be exported to an HDF5 file and also loaded.

Using the model workflow instead of the basic workflow comes with the initial overhead of choosing a suitable combination of parameters and a model class, but has the advantages of a higher level of automation with built-in mechanisms for checking correctness of input (e.g., regarding units), reduced redundancy, and the options to store and load results. Both modes of using the toolbox can also be combined.

2.3 Structure of the Toolbox

The structure of the NNMT repository is depicted in **Figure 1B**. The Python package `nnmt` is subdivided into submodules containing the model classes (`nnmt.models`), helper routines (at present `nnmt.input_output` and `nnmt.utils`), and the tools. The tools are organized in a modular, extensible fashion with a streamlined hierarchy. To give an example, a large part of the currently implemented tools apply to networks of leaky integrate-and-fire (LIF) neurons and they are located in the submodule `nnmt.lif`. The mean-field theory for networks of LIF neurons distinguishes between neurons with instantaneous synapses, also called delta synapses, and those with exponentially decaying post-synaptic currents. Similarly, the submodule for LIF neurons is split further into the two submodules `nnmt.lif.delta` and `nnmt.lif.exp`. NNMT also collects different implementations for computing the same quantity using different approximations or numerics, allowing for a comparison of different approaches.

Apart from the core package, NNMT comes with an extensive online documentation[3], including a quickstart tutorial, all examples presented in this paper, a complete documentation of all tools, as well as a guide for contributors.

Furthermore, we provide an extensive test suite that validates the tools by checking them against previously published results and alternative implementations where possible. This ensures that future improvements of the numerics do not break the tools.

3 HOW TO USE THE TOOLBOX

In this section, we demonstrate the practical use of NNMT by replicating a variety of previously published results. The examples presented have been chosen to cover a broad range of common use cases and network models. We include analyses of both stationary and dynamic network features, as mean-field theory is typically divided into two parts: stationary theory, which describes time-independent network properties of systems in a stationary state, and dynamical theory, which describes time-dependent network properties. Additionally, we show how to use the toolbox to map a spiking to a simpler rate model, as well as how to perform a linear stability analysis.

3.1 Installation and Setup

The toolbox can be either installed using pip

```
pip install nnmt
```

or by installing it directly from the repository, which is described in detail in the online documentation. After the installation, the module can be imported:

```
import nnmt
```

[3] https://nnmt.readthedocs.io/

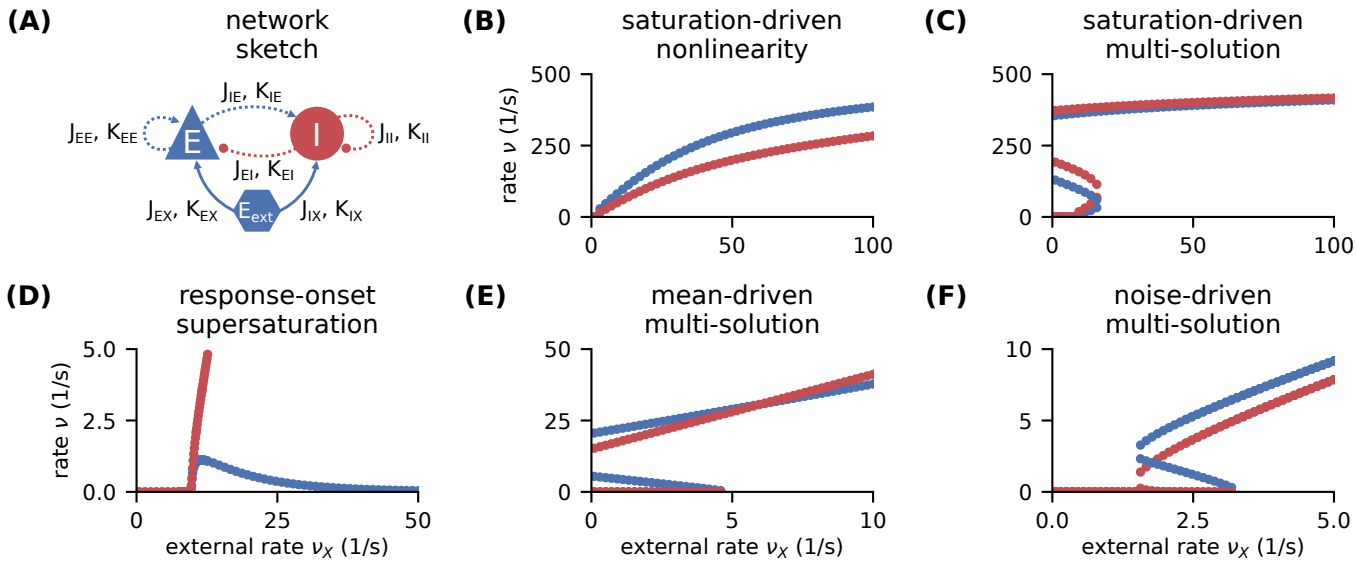

Figure 2. Response nonlinearities in EI-networks. **(A)** Network diagram with nodes and edges according to the graphical notation proposed by Senk et al. (2021). **(B–F)** Firing rate of excitatory (blue) and inhibitory (red) population for varying external input rate ν_X. Specific choices for synaptic weights ($\boldsymbol{J}$, $\boldsymbol{J}_{\text{ext}}$) and in-degrees ($\boldsymbol{K}$, $\boldsymbol{K}_{\text{ext}}$) lead to five types of nonlinearities: **(B)** saturation-driven nonlinearity, **(C)** saturation-driven multi-solution, **(D)** response-onset supersaturation, **(E)** mean-driven multi-solution, and **(F)** noise-driven multi-solution. See Figure 8 in (Sanzeni et al., 2020) for parameters.

3.2 Stationary Quantities

3.2.1 Response Nonlinearities

Networks of excitatory and inhibitory neurons (**Figure 2A**) are widely used in computational neuroscience (Gerstner et al., 2014), e.g., to show analytically that a balanced state featuring asynchronous, irregular activity emerges dynamically in a broad region of the parameter space (Brunel, 2000). How do these networks respond to external input? Sanzeni et al. (2020) uncover five different types of nonlinearities in the network response depending on the network parameters. Here, we show how to use the toolbox to reproduce their result (**Figure 2B–F**).

The network consists of two populations of identical LIF neurons with instantaneous (*delta*) synapses. The synaptic weights and mean in-degrees of recurrent and external connections are population-specific:

$$\boldsymbol{J} = \begin{pmatrix} J_{\text{EE}} & -J_{\text{EI}} \\ J_{\text{IE}} & -J_{\text{II}} \end{pmatrix}, \ \boldsymbol{J}_{\text{ext}} = \begin{pmatrix} J_{\text{EX}} \\ J_{\text{IX}} \end{pmatrix}, \ \boldsymbol{K} = \begin{pmatrix} K_{\text{EE}} & K_{\text{EI}} \\ K_{\text{IE}} & K_{\text{II}} \end{pmatrix}, \text{ and } \boldsymbol{K}_{\text{ext}} = \begin{pmatrix} K_{\text{EX}} \\ K_{\text{IX}} \end{pmatrix} \quad . \tag{1}$$

All weights are positive, implying an excitatory external input. Approximating the input to a neuron as Gaussian white noise $\xi(t)$ with mean $\langle \xi(t) \rangle = \mu$ and noise intensity $\langle \xi(t)\xi(t') \rangle = \tau_{\text{m}}\sigma^2\delta(t-t')$, its firing rate is given by (Siegert, 1951; Amit and Brunel, 1997b)

$$\phi(\mu,\sigma) = \left(\tau_{\text{r}} + \tau_{\text{m}}\sqrt{\pi} \int_{\widetilde{V}_0(\mu,\sigma)}^{\widetilde{V}_{\text{th}}(\mu,\sigma)} e^{s^2}(1+\text{erf}(s))\text{d}s \right)^{-1} \quad , \tag{2}$$

where τ_m denotes the membrane time constant, τ_r the refractory period, and the rescaled reset- and threshold-voltages are

$$\widetilde{V}_0(\mu,\sigma) = \frac{V_0-\mu}{\sigma} \quad , \qquad \widetilde{V}_\mathrm{th}(\mu,\sigma) = \frac{V_\mathrm{th}-\mu}{\sigma} \quad . \tag{3}$$

The first term in Eq. (2) is the refractory period and the second term is the mean first-passage time of the membrane voltage from reset to threshold. The mean and the noise intensity of the input to a neuron in a population A, which control the mean first-passage time through Eq. (3), are determined by (Amit and Brunel, 1997b)

$$\mu_A = \tau_\mathrm{m}(J_{A\mathrm{E}}K_{A\mathrm{E}}\nu_\mathrm{E} - J_{A\mathrm{I}}K_{A\mathrm{I}}\nu_\mathrm{I} + J_{A\mathrm{X}}K_{A\mathrm{X}}\nu_\mathrm{X}) \quad , \tag{4}$$

$$\sigma_A^2 = \tau_\mathrm{m}(J_{A\mathrm{E}}^2K_{A\mathrm{E}}\nu_\mathrm{E} + J_{A\mathrm{I}}^2K_{A\mathrm{I}}\nu_\mathrm{I} + J_{A\mathrm{X}}^2K_{A\mathrm{X}}\nu_\mathrm{X}) \quad , \tag{5}$$

respectively, where each term reflects the contribution of one population, with the corresponding firing rates of the excitatory ν_E, inhibitory ν_I, and external population ν_X. Note that the excitatory and inhibitory inputs counteract each other in μ_A, Eq. (4), while their contributions add up in σ_A, Eq. (5). Both μ_A and σ_A depend on the firing rate of the neurons ν_A, which is in turn given by Eq. (2). Thus, one arrives at the self-consistency problem

$$\nu_A = \phi(\mu_A,\sigma_A) \quad , \tag{6}$$

which is coupled across the populations due to Eq. (4) and Eq. (5).

Our toolbox provides two algorithms to solve Eq. (6): 1) Integrating the auxiliary ordinary differential equation (ODE) $\dot{\nu}_A = -\nu_A + \phi(\mu_A,\sigma_A)$ with initial values $\nu_A(0) = \nu_{A,0}$ until it reaches a fixed point $\dot{\nu}_A = 0$, where Eq. (6) holds by construction. 2) Minimizing the quadratic deviation $\sum_A \left[\nu_A - \phi(\mu_A,\sigma_A)\right]^2$, using least squares (LSTSQ) starting from an initial guess $\nu_{A,0}$. The ODE method is robust to changes in the initial values and hence a good first choice. However, it cannot find self-consistent solutions that correspond to an unstable fixed point of the auxiliary ODE. To this end, the LSTSQ method can be used. Its drawback is that it needs a good initial guess, because otherwise the found minimum might be a local one where the quadratic deviation does not vanish and which does not correspond to a self-consistent solution. A prerequisite for both methods is a numerical solution of the integral in Eq. (2); this is discussed in the Appendix, Section 5.1.

The solutions of the self-consistency problem Eq. (6) for varying ν_X and fixed $\boldsymbol{J}$, $\boldsymbol{J}_\mathrm{ext}$, $\boldsymbol{K}$, and $\boldsymbol{K}_\mathrm{ext}$ reveal the five types of response nonlinearities (**Figure 2**). Different response nonlinearities arise through specific choices of synaptic weights, $\boldsymbol{J}$ and $\boldsymbol{J}_\mathrm{ext}$, and in-degrees, $\boldsymbol{K}$ and $\boldsymbol{K}_\mathrm{ext}$, which suggests that already a simple EI-network possesses a rich capacity for nonlinear computations. Whenever possible, we use the ODE method and resort to the LSTSQ method only if the self-consistent solution corresponds to an unstable fixed point of the auxiliary ODE. Combining both methods, we can reproduce the first columns of Figure 8 in Sanzeni et al. (2020), where all five types of nonlinearities are presented. Curiously, some of the solutions corresponding to an unstable fixed point are nonetheless found in network simulations according to other panels of Figure 8 in Sanzeni et al. (2020) which we do not reproduce here.

In **Listing 2**, we show a minimal example to produce the data shown in **Figure 2B**. After importing the function that solves the self-consistency Eq. (6), we collect the neuron and network parameters in a dictionary. Then, we loop through different values for the external rate ν_X and determine the network

```python
import numpy as np
from nnmt.lif.delta import _firing_rates

params = dict(
    # membrane and refractory time constants (in s)
    tau_m=20.*1e-3, tau_r=2.*1e-3,
    # relative reset and threshold potentials (in V)
    V_0_rel=10.*1e-3, V_th_rel=20.*1e-3,
    # recurrent and external weights (in V)
    J=np.array([[0.2, -1.6], [0.2, -1.4]])*1e-3,
    J_ext=np.array([0.2, 0.2])*1e-3,
    # recurrent and external in-degrees
    K=np.array([[400, 100], [400, 100]]),
    K_ext=np.array([1600, 800]))

# determine self-consistent rates (in 1/s)
nu_ext = np.linspace(1, 100, 50) # external rates (in 1/s)
nu_E, nu_I = np.zeros_like(nu_ext), np.zeros_like(nu_ext)
for i, nu_X in enumerate(nu_ext):
    nu_E[i], nu_I[i] = _firing_rates(nu_ext=nu_X, nu_0=(0, 0),
                                     fixpoint_method='ODE', **params)
```

Listing 2: Example script to produce the data shown in **Figure 2B** using the ODE method (initial value $\nu_{A,0} = 0$ for population $A \in \{E, I\}$).

rates using the ODE method, which is sufficient in this example. In **Listing 2** and to produce **Figure 2**, we use the basic workflow because only one isolated tool of NNMT (`nnmt.lif.delta._firing_rates()`) is employed, which requires only a few parameters defining the simple EI-network.

3.2.2 Firing Rates of Microcircuit Model

Here we show how to use the model workflow to calculate the firing rates of the cortical microcircuit model by Potjans and Diesmann (2014). The circuit is a simplified point neuron network model with biologically plausible parameters which has been recently used in a number of other works, for example to study network properties, to assess the performance of different simulators, and to extend it to large-scale models (e.g., Wagatsuma et al., 2011; Bos et al., 2016; Hagen et al., 2016; Schmidt et al., 2018; van Albada et al., 2018; Knight and Nowotny, 2018; Golosio et al., 2021; Dasbach et al., 2021). The model consists of eight populations of LIF neurons, corresponding to the excitatory and inhibitory populations of four cortical layers: 2/3E, 2/3I, 4E, 4I, 5E, 5I, 6E, and 6I (see **Figure 3A**). It defines the number of neurons in each population, the number of connections between the populations, the single neuron properties, and the external input. Simulations show that the model yields realistic firing rates for the different populations (Potjans and Diesmann, 2014).

In contrast to the EI-network model investigated in Section 3.2.1, the neurons in the microcircuit model have exponentially shaped post-synaptic currents. Fourcaud and Brunel (2002) developed a method for calculating the firing rate for this synapse type. They have shown that, if the synaptic time constant τ_s is much smaller than the membrane time constant τ_m, the firing rate for LIF neurons with exponential synapses can be calculated using Eq. (2) with shifted integration boundaries

$$\widetilde{V}_{\mathrm{cn},0}(\mu,\sigma) = \widetilde{V}_0(\mu,\sigma) + \frac{\alpha}{2}\sqrt{\frac{\tau_\mathrm{s}}{\tau_\mathrm{m}}} \quad , \qquad \widetilde{V}_{\mathrm{cn,th}}(\mu,\sigma) = \widetilde{V}_\mathrm{th}(\mu,\sigma) + \frac{\alpha}{2}\sqrt{\frac{\tau_\mathrm{s}}{\tau_\mathrm{m}}} \quad , \tag{7}$$

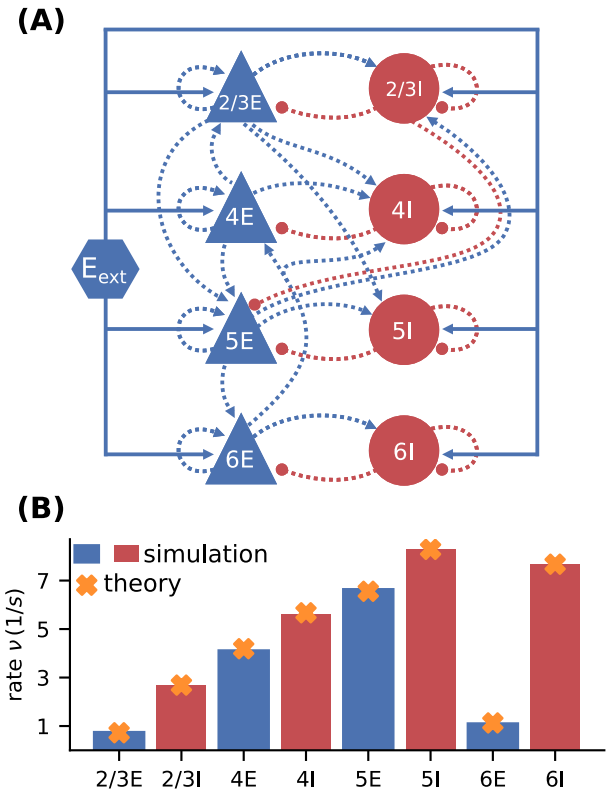

Figure 3. Cortical microcircuit model by Potjans and Diesmann (2014). **(A)** Network diagram (only the strongest connections are shown as in Figure 1 of the original publication). Same notation as in **Figure 2A**. **(B)** Simulation and mean-field estimate for average population firing rates using the parameters from Bos et al. (2016).

with the rescaled reset- and threshold-voltages from Eq. (3) and $\alpha = \sqrt{2}\,|\zeta(1/2)|$, where $\zeta(x)$ denotes the Riemann zeta function; the subscript cn stands for "colored noise".

The microcircuit has been implemented as an NNMT model (`nnmt.models.Microcircuit`). We here use the parameters of the circuit as published in Bos et al. (2016) which is slightly differently parameterized than the original model (see Appendix, **Table 1**). The model parameter values and units are defined using a `yaml` file, which uses a dictionary style syntax with `key:value` pairs:

```yaml
# membrane time constant
tau_m:
  val: 10.0
  unit: ms

# neuron numbers
N:
  - 20683
  - 5834
  - 21915
```

Once the parameters are defined, a microcircuit model is instantiated by passing the respective parameter file to the model constructor; the units are automatically converted to SI units. Then the firing rates are computed. For comparison, we finally load the simulated rates from Bos et al. (2016):

```
# create the network model using a network parameter yaml file
microcircuit = nnmt.models.Microcircuit('network_params_microcircuit.yaml')
# calculate firing rates
firing_rates = nnmt.lif.exp.firing_rates(microcircuit)
# load simulated results
simulated_firing_rates = \
    nnmt.input_output.load_h5('Bos2016_rates.h5')['rates']
```

The simulated rates have been obtained by a numerical network simulation in which the neuron populations are connected according to the model's original connectivity rule: random, fixed total number with multapses (autapses prohibited), see Senk et al. (2021). For simplicity, the theoretical predictions assume a connectivity with a fixed in-degree for each neuron. Dasbach et al. (2021) show that simulated spike activity data of networks with these two different connectivity rules are characterized by differently shaped rate distributions ("reference" in their Figures 3d and 4d). In addition, the weights in the simulation are normally distributed while the theory replaces each distribution by its mean; this corresponds to the case $N_{\text{bins}} = 1$ in Dasbach et al. (2021). Nevertheless, our mean-field theoretical estimate of the average population firing rates is in good agreement with the simulated rates (**Figure 3B**).

3.3 Dynamical Quantities

3.3.1 Transfer Function

One of the most important dynamical properties of a neuronal network is how it reacts to external input. A systematic way to study the network response is to apply an oscillatory external input with fixed frequency ω, phase, and amplitude and observe the emerging frequency, phase, and amplitude of the output. If the amplitude of the external input is small compared to the stationary input, the network responds in a linear fashion: it only modifies phase and amplitude, while the output frequency equals the input frequency. This relationship is captured by the input-output transfer function $N(\omega)$ (Brunel et al., 2001; Brunel and Hakim, 1999; Lindner and Schimansky-Geier, 2001) which describes the frequency-dependent modulation of the output firing rate

$$\nu(t) = \nu + N(\omega)\,\delta\mu\,\mathrm{e}^{\mathrm{i}\omega t} \quad ,$$

where $\delta\mu$ is the amplitude of the modulation to the input in units of Volt. The transfer function $N(\omega)$ is a complex function: Its absolute value describes the relative modulation of the firing rate. Its phase describes the phase shift that occurs between input and output. Since response properties are dominated by the modulation to the mean input, we here disregard a modulation of the noise intensity and denote the transfer function for a network of LIF neurons with instantaneous synapses as

$$N(\omega) = \frac{\sqrt{2}\nu}{\sigma}\,\frac{1}{1+\mathrm{i}\omega\tau_{\mathrm{m}}}\,\frac{\Phi'_\omega\big|_{\sqrt{2}\widetilde{V}_0}^{\sqrt{2}\widetilde{V}_{\mathrm{th}}}}{\Phi_\omega\big|_{\sqrt{2}\widetilde{V}_0}^{\sqrt{2}\widetilde{V}_{\mathrm{th}}}} \quad , \tag{8}$$

with the rescaled reset- and threshold-voltages $\widetilde{V}_0$ and $\widetilde{V}_{\mathrm{th}}$ as defined in Eq. (3) and $\Phi_\omega(x) = e^{\frac{x^2}{4}}\,U\left(\mathrm{i}\omega\tau_{\mathrm{m}} - \frac{1}{2}, x\right)$ using the parabolic cylinder functions $U\left(\mathrm{i}\omega\tau_{\mathrm{m}} - \frac{1}{2}, x\right)$ as defined in (Abramowitz and Stegun, 1974, Section 19.3) and (Olver et al., 2021, Section 12.2). Φ'_ω and Φ''_ω denote

the first and second derivative by x, respectively. The transfer function given in Schuecker et al. (2015, Eq. 29) relates to our expression in Eq. (8) as follows: Our $N(\omega)\,\delta\mu$ is equal to their $n_G(\omega)\,\nu$ implying that our $\delta\mu$ corresponds to their $\epsilon\mu$; we assume their $n_H(\omega) = 0$. Besides, we swap the voltage boundaries both in numerator and denominator which cancels out.

For a neuronal network of LIF neurons with exponentially shaped post-synaptic currents, Schuecker et al. (2014, 2015) show that an analytical approximation of the colored-noise transfer function can be obtained by a shift of integration boundaries akin to Eq. (7):

$$N_{\mathrm{cn}}(\omega) = \frac{\sqrt{2}\nu}{\sigma} \frac{1}{1 + \mathrm{i}\omega\tau_{\mathrm{m}}} \frac{\Phi'_\omega|_{\sqrt{2}\widetilde{V}_{\mathrm{cn,0}}}^{\sqrt{2}\widetilde{V}_{\mathrm{cn,th}}}}{\Phi_\omega|_{\sqrt{2}\widetilde{V}_{\mathrm{cn,0}}}^{\sqrt{2}\widetilde{V}_{\mathrm{cn,th}}}} \quad . \tag{9}$$

To take into account that, in neuronal networks, the current in Ampere is perturbed by the synaptic input, we include an additional low-pass filtering with the synaptic time constant into the definition:

$$N_{\mathrm{cn,s}}(\omega) = N_{\mathrm{cn}}(\omega) \frac{1}{1 + \mathrm{i}\omega\tau_{\mathrm{s}}} \quad . \tag{10}$$

If the synaptic time constant is much smaller than the membrane time constant ($\tau_{\mathrm{s}} \ll \tau_{\mathrm{m}}$), an equivalent expression for the transfer function is obtained by a Taylor expansion around the original boundaries (cf. Schuecker et al. 2015, Eq. 30). The toolbox implements both variants and offers choosing between them by setting the argument `method` of `nnmt.lif.exp.transfer_function` to either `shift` or `taylor`.

Here, we demonstrate how to calculate the analytical "shift version" of the transfer function for different means and noise intensities of the input current (see **Figure 4**) and thereby reproduce Figure 4 in Schuecker et al. (2015).

The crucial parts for producing **Figure 4** using NNMT are shown in **Listing 3** for one example combination of mean and noise intensity of the input current. Instead of using the model workflow with `nnmt.lif.exp.transfer_function`, we here employ the basic workflow, using `nnmt.lif.exp._transfer_function_shift` directly. This allows changing the mean input and its noise intensity independently of a network model's structure, but requires two additional steps: First, the necessary parameters are loaded from a `yaml` file, converted to SI units and then stripped off the units using the utility function `nnmt.utils._convert_to_si_and_strip_units`. Second, the analysis frequencies are defined manually. In this example we choose logarithmically spaced frequencies, as we want to plot the results on a log-scale. Finally, the complex-valued transfer function is calculated and then split into its absolute value and phase. **Figure 4** shows that the transfer function acts as a low-pass filter that suppresses the amplitude of high frequency activity, introduces a phase lag, and can lead to resonance phenomena for certain configurations of mean input current and noise intensity.

The replication of the results from Schuecker et al. (2015) outlined here is also used in the integration tests of toolbox. Note that the implemented analytical form of the transfer function by Schuecker et al. (2015) is an approximation and deviations from a simulated ground truth are expected for higher frequencies ($\omega/2\pi \gtrsim 100\,\mathrm{Hz}$ at the given parameters).

```python
# load parameters in custom units
params = nnmt.input_output.load_val_unit_dict_from_yaml(
    'Schuecker2015_parameters.yaml')

# convert parameters to SI units
nnmt.utils._convert_to_si_and_strip_units(params)

# define the analysis frequencies
frequencies = np.logspace(
    params['f_start_exponent'],
    params['f_end_exponent'],
    params['n_freqs'])
# add the zero frequency
frequencies = np.insert(frequencies, 0, 0.0)
omegas = 2 * np.pi * frequencies

# extract necessary parameters from params dictionary
mean_input = params['mean_input']
...  # here we leave out similar statements

# calculate the transfer function using parameters in SI units
transfer_function = nnmt.lif.exp._transfer_function_shift(
    mu, sigma,
    tau_m, tau_s, tau_r,
    V_th_rel, V_0_rel,
    omegas,
    synaptic_filter=False)

# calculate properties plotted in Schuecker et al. (2015)
absolute_value = np.abs(transfer_function)
phase = np.angle(transfer_function) / 2 / np.pi * 360
```

Listing 3: Example script for computing a transfer function shown in **Figure 4** using the method of shifted integration boundaries.

3.3.2 Power Spectrum

Another dynamical property that is studied frequently is the power spectrum, which describes how the power of a signal is distributed across its different frequency components. This property for example reveals oscillations of the population activity. The power can be read off the diagonal entries of the cross-correlations between population activities in the Fourier domain. The cross-correlations are determined by the network architecture, the stationary firing rates, and the network's transfer function. The mean-field approximation of the power spectrum is obtained by studying the linear response of the network activity to the fluctuations caused by its spiking activity (Bos et al., 2016). The resulting analytical expression for the power spectra at a given angular frequency ω is

$$\mathbf{C}(\omega) = \left(\mathbf{1} - \widetilde{\mathbf{M}}_{\mathrm{d}}(\omega)\right)^{-1} \mathrm{diag}\left(\boldsymbol{\nu} \oslash \boldsymbol{n}\right) \left(\mathbf{1} - \widetilde{\mathbf{M}}_{\mathrm{d}}(-\omega)\right)^{-\mathrm{T}} \quad , \tag{11}$$

with $\oslash$ denoting the elementwise (Hadamard) division, the effective connectivity matrix $\widetilde{\mathbf{M}}_d(\omega) = \tau_{\mathrm{m}} \boldsymbol{JK} \left[\boldsymbol{N}_{\mathrm{cn,s}}(\omega)^{\mathrm{T}} \odot \boldsymbol{D}(\omega)\right]$, where $\odot$ denotes the elementwise (Hadamard) product, the mean population firing rates $\boldsymbol{\nu}$, and the numbers of neurons in each population $\boldsymbol{n}$. This effective connectivity combines the static, anatomical connectivity $\boldsymbol{JK}$, represented by synaptic weight matrix $\boldsymbol{J}$ and in-degree matrix $\boldsymbol{K}$, and

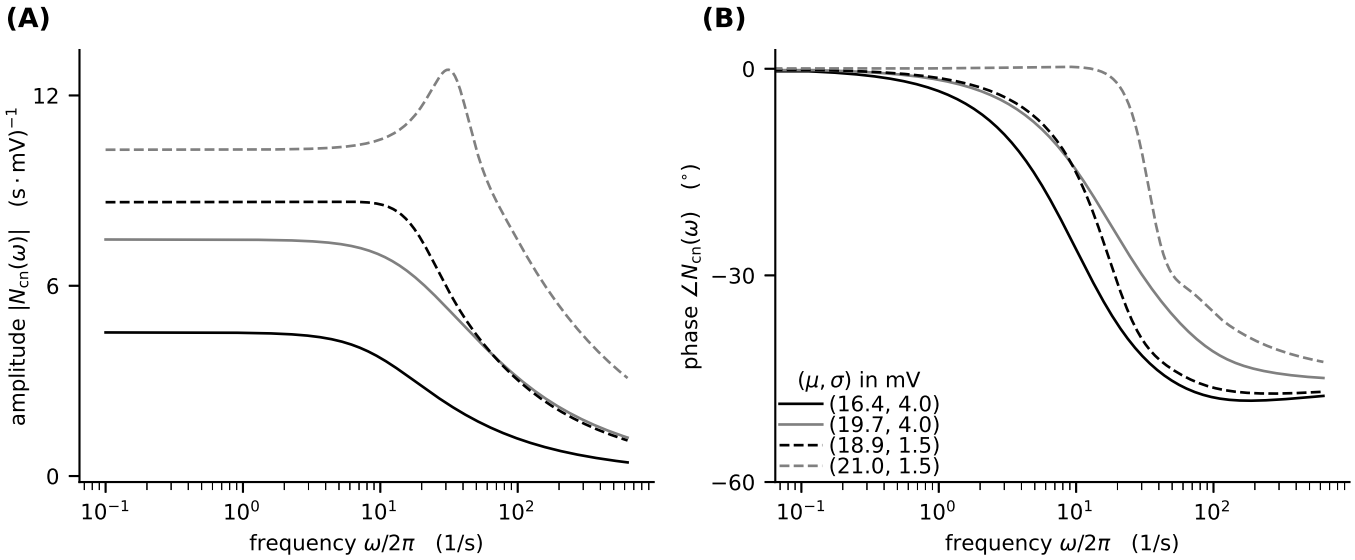

Figure 4. Colored-noise transfer function N_{cn} of LIF model in different regimes. **(A)** Absolute value and **(B)** phase of the "shift" version of the transfer function as a function of the log-scaled frequency. Neuron parameters are set to $V_{\mathrm{th}} = 20\,\mathrm{mV}$, $V_0 = 15\,\mathrm{mv}$, $\tau_{\mathrm{m}} = 20\,\mathrm{ms}$, and $\tau_{\mathrm{s}} = 0.5\,\mathrm{ms}$. For given noise intensities of input current, $\sigma = 4\,\mathrm{mV}$ (solid line) and $\sigma = 1.5\,\mathrm{mV}$ (dashed line), the mean input μ is chosen such that firing rates $\nu = 10\,\mathrm{Hz}$ (black) and $\nu = 30\,\mathrm{Hz}$ (gray) are obtained.

dynamical quantities, represented by the transfer functions $\boldsymbol{N}_{\mathrm{cn,s}}(\omega)$ (Eq. (10)) and the delay distributions $\boldsymbol{D}(\omega)$, both in Fourier domain.

The modular structure in combination with the model workflow of this toolbox permits a step-by-step calculation of the power spectra, as shown in **Listing 4**. The inherent structure of the theory is emphasized in these steps: After instantiating the network model class with given network parameters, the working point can be calculated, which includes calculating the population firing rates and the input statistics. This is necessary for determining the transfer functions. The calculation of the delay distribution matrix is then required for calculating the effective connectivity and to finally get an estimate of the power spectra. **Figure 5** reproduces Figure 1E in Bos et al. (2016) and shows the spectra for each population of the adjusted version (see **Table 1**) of the microcircuit model.

The numerical predictions obtained from the toolbox coincide with simulated data taken from the original publication (Bos et al., 2016) and reveal dominant oscillations of the population activities in the low-γ range around $64\,\mathrm{Hz}$. Furthermore, faster oscillations with peak power around $300\,\mathrm{Hz}$ are predicted with higher magnitudes in the inhibitory populations 4I, 5I, and 6I.

3.3.3 Sensitivity Measure

The power spectra shown in the previous section exhibit prominent peaks at certain frequencies, which indicate oscillatory activity. Naturally, this begs the question: which mechanism causes these oscillations? Bos et al. (2016) expose the crucial role that the microcircuit's connectivity plays in shaping the power spectra of this network model. They have developed a method called *sensitivity measure* to directly relate the influence of the anatomical connections, especially the in-degree matrix, on the power spectra.

The power spectrum of the i-th population $C_{ii}(\omega)$ receives a contribution from each eigenvalue λ_j of the effective connectivity matrix, $C_{ii}(\omega) \propto 1/\left(1 - \lambda_j(\omega)\right)^2$. Such a contribution consequently diverges as the complex-valued λ_j approaches $1 + 0\mathrm{i}$ in the complex plane, which is referred to as the point of

```python
# create network model microcircuit
microcircuit = nnmt.models.Microcircuit(
    network_params='Bos2016_network_params.yaml',
    analysis_params='Bos2016_analysis_params.yaml')

# calculate working point for exponentially shaped post-synaptic currents
nnmt.lif.exp.working_point(microcircuit, method='taylor')
# calculate the transfer function
nnmt.lif.exp.transfer_function(microcircuit, method='taylor')
# calculate the delay distribution matrix
nnmt.network_properties.delay_dist_matrix(microcircuit)
# calculate the effective connectivity matrix
nnmt.lif.exp.effective_connectivity(microcircuit)
# calculate the power spectra
power_spectra = nnmt.lif.exp.power_spectra(microcircuit).
```

Listing 4: Example script to produce the data shown in **Figure 5**.

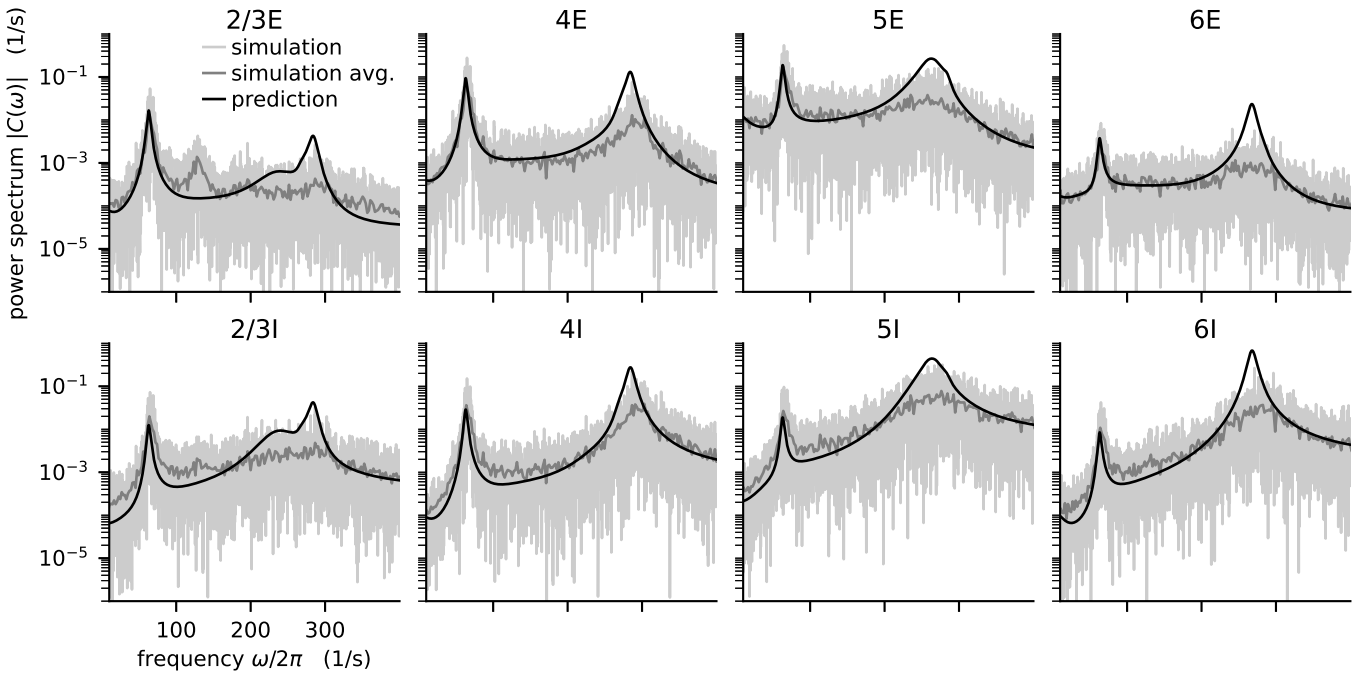

Figure 5. Power spectra of the population spiking activity in the adapted cortical microcircuit from Bos et al. (2016). The spiking activity of each population in a 10 s simulation of the model is binned with 1 ms resolution and the power spectrum of the resulting histogram is calculated by a fast Fourier transform (FFT; light gray curves). In addition, the simulation is split into 500 ms windows, the power spectrum calculated for each window and averaged across windows (gray curves). Black curves correspond to analytical prediction obtained with NNMT as described in **Listing 4**.

instability. This relation can be derived by replacing the effective connectivity matrix $\widetilde{\mathbf{M}}_d(\omega)$ in Eq. (11) by its eigendecomposition. The sensitivity measure leverages this relationship and evaluates how a change in the in-degree matrix affects the eigenvalues of the effective connectivity and thus indirectly the power spectrum. Bos et al. (2016) introduce a small perturbation α_{kl} of the in-degree matrix, which allows writing the effective connectivity matrix as $\widetilde{M}_{ij}(\omega) = \left(1 + \alpha_{kl}\delta_{ki}\delta_{lj}\right) \widetilde{M}_{ij}(\omega)$, where we dropped the

delay subscript d. The sensitivity measure $Z_{j,kl}(\omega)$ describes how the j-th eigenvalue of the effective connectivity matrix varies when the kl-th element of the in-degree matrix is changed

$$Z_{j,kl}(\omega) = \frac{\partial \lambda_j(\omega)}{\partial \alpha_{kl}}\bigg|_{\alpha_{kl}=0} = \frac{v_{j,k}\widetilde{M}_{kl}u_{j,l}}{\mathbf{v}_j^{\mathrm{T}} \cdot \mathbf{u}_j} \quad , \tag{12}$$

where $\mathbf{v}_j^{\mathrm{T}}$ and $\mathbf{u}_j$ are the left and right eigenvectors of $\widetilde{\boldsymbol{M}}$ corresponding to eigenvalue $\lambda_j(\omega)$.

The complex sensitivity measure can be understood in terms of two components: $\mathbf{Z}_j^{\mathrm{amp}}$ is the projection of the matrix $\mathbf{Z}_j$ onto the direction in the complex plane defined by $1 - \lambda_j(\omega)$; it describes how, when the in-degree matrix is perturbed, the complex-valued $\lambda_j(\omega)$ moves towards or away from the instability $1 + 0i$, and consequently how the amplitude of the power spectrum at frequency ω increases or decreases. $\mathbf{Z}_j^{\mathrm{freq}}$ is the projection onto the perpendicular direction and thus describes how the peak frequency of the power spectrum changes with the perturbation of the in-degree matrix.

The toolbox makes this intricate measure accessible by supplying two tools: After computing the required working point, transfer function, and delay distribution, the tool `nnmt.lif.exp.sensitivity_measure` computes the sensitivity measure at a given frequency for one specific eigenvalue. By default, this is the eigenvalue which is closest to the instability $1 + 0i$. To perform the computation, we just need to add one line to **Listing 4**:

```
sensitivity_dict = nnmt.lif.exp.sensitivity_measure(microcircuit, frequency)
```

The result is returned in form of a dictionary that contains the sensitivity measure and its projections. The tool `nnmt.lif.exp.sensitivity_measure_per_eigenmode` wraps that basic function and calculates the sensitivity measure for all eigenvalues at the frequency for which each eigenvalue is closest to instability.

According to the original publication (Bos et al., 2016), the peak around 300 Hz has contributions from four different eigenvalues of the effective connectivity matrix. **Figure 6** shows the projections of the sensitivity measure at the frequency for which the corresponding eigenvalues are closest to the instability, as in Figure 4 of Bos et al. (2016). The sensitivity measure returns one value for each connection between populations in the network model. For $\mathbf{Z}_j^{\mathrm{amp}}$ a negative value indicates that increasing the in-degrees of a specific connection causes the amplitude of the power spectrum at the evaluated frequency to drop. If the value is positive, the amplitude is predicted to grow as the in-degrees increase. Similarly, for positive $\mathbf{Z}_j^{\mathrm{freq}}$ the frequency of the peak in the power spectrum shifts towards higher values as in-degrees increase, and vice versa. The main finding in this analysis is that the high-γ peak seems to be affected by inhibitory self-connections of each population.

To decrease the high-γ peak in the power spectrum, one could therefore increase the 4I to 4I connections (cp. panel **(A)** in **Figure 6**)

```
K_new = microcircuit.network_params['K'].copy()
K_new[3,3] = 1000.0 # originally 953
microcircuit_new = microcircuit.change_parameters({'K': K_new})
```

and calculate the power spectrum as in **Listing 4** again to validate the change.

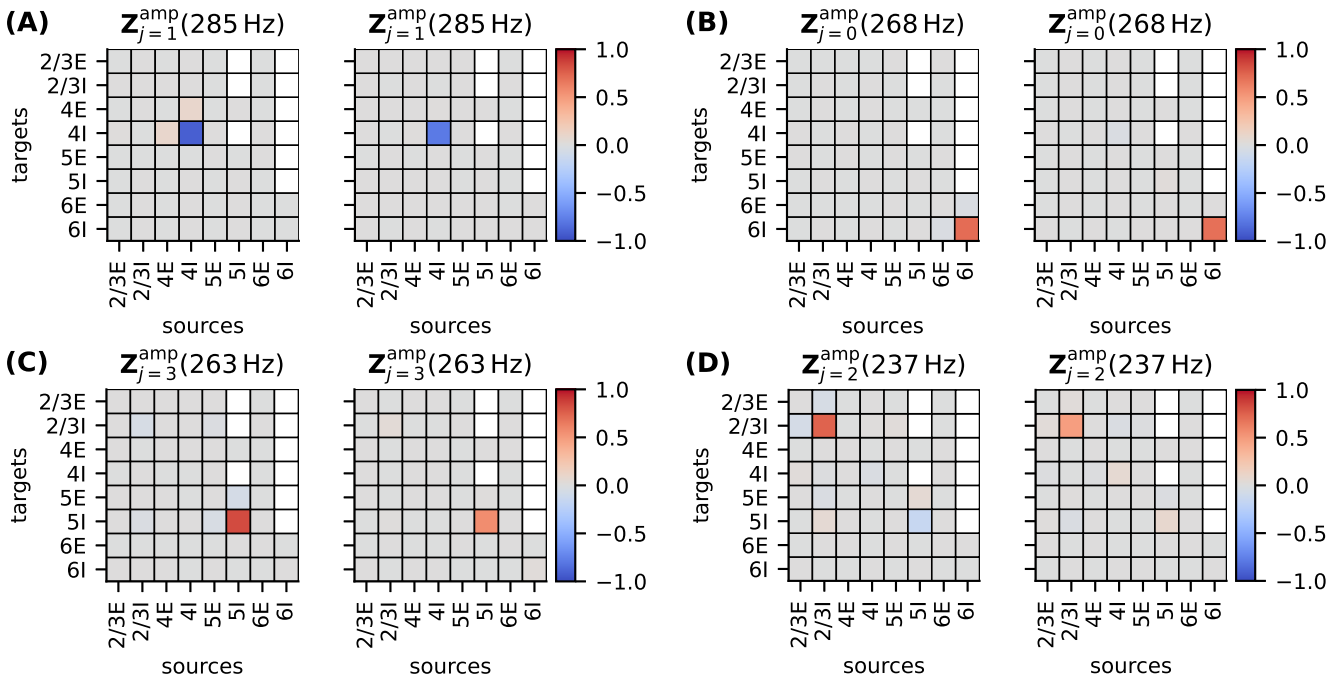

Figure 6. Sensitivity measure of four eigenmodes of the effective connectivity relevant for high-γ oscillations. The sensitivity measure for each mode is evaluated at the frequency where the corresponding eigenvalue is closest to the point of instability $1 + 0i$ in complex plane. $\mathbf{Z}_j^{\mathrm{amp}}(\omega)$ visualizes the influence of a perturbation of a connection on the peak amplitude of the power spectrum. $\mathbf{Z}_j^{\mathrm{freq}}(\omega)$ shows the impact on the peak frequency. Non-existent connections are masked white.

3.4 Fitting Spiking to Rate Model and Predicting Pattern Formation

If the neurons of a network are spatially organized and connected according to a distance-dependent profile, the spiking activity may exhibit pattern formation in space and time, including wave-like phenomena. Senk et al. (2020) set out to scrutinize the non-trivial relationship between the parameters of such a network model and the emerging activity patterns. The model they use is a two-population network of excitatory E and inhibitory I spiking neurons, illustrated in **Figure 7**. All neurons are of type LIF with exponentially shaped post-synaptic currents. The neuron populations are recurrently connected to each other and themselves and they receive additional external excitatory $\mathrm{E_{ext}}$ and inhibitory $\mathrm{I_{ext}}$ Poisson spike input of adjustable rate as shown in **Figure 7A**. The spatial arrangement of neurons on a ring is illustrated in **Figure 7B** and the boxcar-shaped connectivity profiles in **Figure 7C**.

In the following, we consider a mean-field approximation of the spiking model with spatial averaging, that is a time and space continuous approximation of the discrete model as derived in Senk et al. (2020, E. Linearization of spiking network model). We demonstrate three methods used in the original study: First, Section 3.4.1 explains how a model can be brought to a defined state characterized by its working point. The working point is given by the mean μ and noise intensity σ of the input to a neuron, which are both quantities derived from network parameters and require the calculation of the firing rates. With the spiking model in that defined state, Section 3.4.2 then maps its transfer function to the one of a rate model. Section 3.4.3 finally shows that this working-point dependent rate model allows for an analytical linear stability analysis of the network accounting for its spatial structure. This analysis can reveal transitions to spatial and temporal oscillatory states which, when mapped back to the parameters of the spiking model, may manifest in distinct patterns of simulated spiking activity.

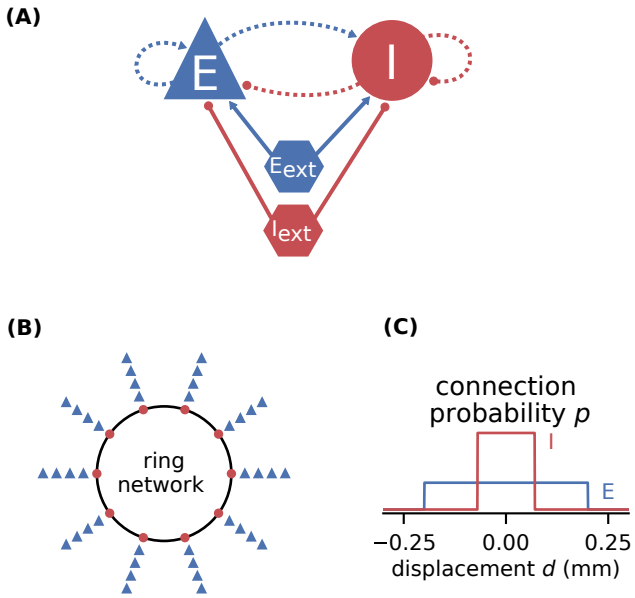

Figure 7. Illustrations of spiking network model by Senk et al. (2020). **(A)** Excitatory and inhibitory neuronal populations randomly connected with fixed in-degree and multapses allowed (autapses prohibited). External excitatory and inhibitory Poisson drive to all neurons. Same notation as in **Figure 2A**. **(B)** One inhibitory and three excitatory neurons per grid position on a one-dimensional domain with periodic boundary conditions (ring network). **(C)** Normalized, boxcar-shaped connection probability with wider excitation than inhibition. For model details and parameters, see Tables II–IV of Senk et al. (2020); the specific values given in the caption of their Figure 6 are used throughout here.

3.4.1 Setting the working point by changing network parameters

With network and analysis parameters predefined in `yaml` files, we set up a network model using the example model class `Basic`:

```
space_model = nnmt.model.Basic(
    network_params='Senk2020_network_params.yaml',
    analysis_params='Senk2020_analysis_params.yaml')
```

Upon initialization the given parameters are automatically converted into the format used by NNMT's tools. For instance, relative spike reset and threshold potentials are derived from the absolute values, connection strengths in units of volt are computed from the post-synaptic current amplitudes in ampere, and all values are scaled to SI units.

We aim to bring the network to a defined state by fixing the working point but also want to explore if the procedure of fitting the transfer function still works for different network states. For a parameter space exploration, we use a method to change parameters provided by the model class and scan through a number of different working points of the network. To obtain the required input for a target working point, we adjust the external excitatory and inhibitory firing rates accordingly; NNMT uses a vectorized version of the equations given in Senk et al. (2020, Appendix F) to calculate the external rates needed:

```
mu = 10. * 1e-3; sigma = 10. * 1e-3 # relative to spike threshold (in V)
nu_ext = nnmt.lif.exp.external_rates_for_fixed_input(
    network, mu_set=mu, sigma_set=sigma)
```

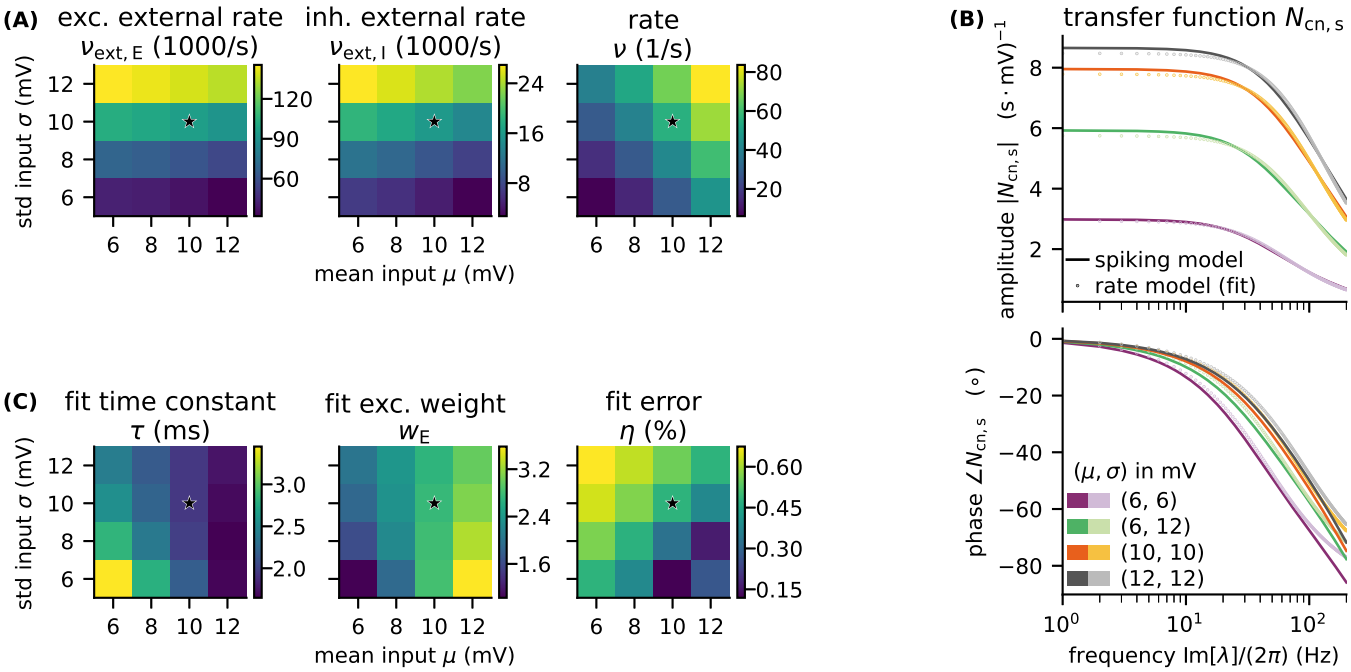

Figure 8. Network parameters and mean-field results from scanning through different working points. Working point $(\mu,\ \sigma)$ combines mean input μ and noise intensity of input σ. **(A)** External excitatory $\nu_{\text{ext, E}}$ and inhibitory $\nu_{\text{ext, I}}$ Poisson rates required to set $(\mu,\ \sigma)$ and resulting firing rates ν. **(B)** Transfer function $N_{\text{cn,s}}$ of spiking model and fitted rate-model approximation with low-pass filter for selected $(\mu,\ \sigma)$ (top: amplitude, bottom: phase). **(C)** Fit results (time constants τ and excitatory weights w_{E}) and fit errors η. The inhibitory weights are $w_{\text{I}} = -g w_{\text{E}}$ with $g = 5$. Star marker in panels (A) and (C) denotes target working point $(10,\ 10)$ mV. Similar displays as in Senk et al. (2020, Figure 5).

```
space_model = space_model.change_parameters(
    changed_network_params={'nu_ext': nu_ext})
```

The resulting external rates for different choices of $(\mu,\ \sigma)$ are color-coded in the first two plots of **Figure 8A**. The third plot shows the corresponding firing rates of the neurons, which are stored in the results of the model instance when computing the working point explicitly:

```
nnmt.lif.exp.working_point(network)
```

3.4.2 Parameter mapping by fitting the transfer function

We map the parameters of the spiking model to a corresponding rate model by, first, computing the transfer function $N_{\text{cn,s}}$ given in Eq. (10) of the spiking model, and second, performing a least-squares fit to the simpler transfer function of the rate model, for details see Senk et al. (2020, F Comparison of neural-field and spiking models). The dynamics of the rate model can be written as a convolution equation with the temporal kernel $\Theta(t) \cdot w/\tau \cdot \exp(-t/\tau)$ using the Heaviside function Θ; τ is the time constant and w the unitless weight. Hence, the transfer function is just the one of a low-pass filter, $N_{\text{LP}} = w/(1 + \lambda\tau)$, where λ is the frequency in Laplace domain. The tool to fit the transfer function requires that the actual transfer function has been computed beforehand and returns the fit for the same frequencies together with τ, w, and the combined fit error η:

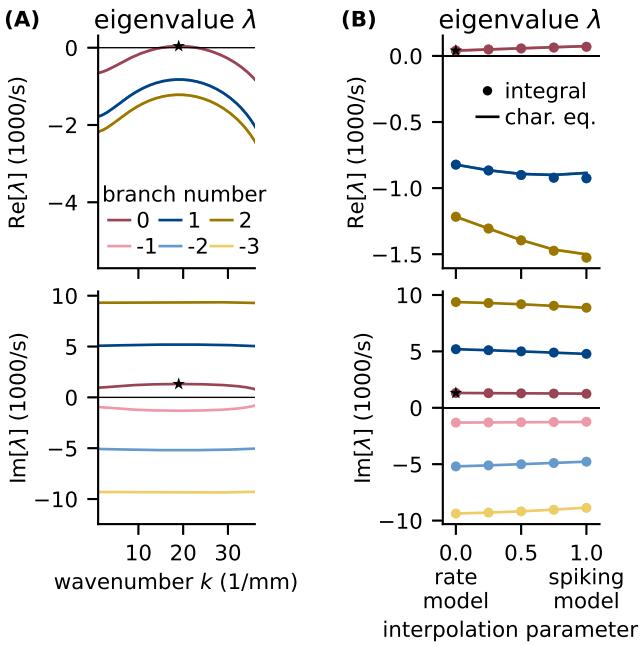

Figure 9. Linear stability analysis of spatially structured network model. **(A)** Analytically exact solution for real (top) and imaginary (bottom) part of eigenvalue λ versus wavenumber k using rate model derived by fit of spiking model at working point $(\mu,\, \sigma) = (10,\, 10)\ \mathrm{mV}$. Color-coded branches of Lambert W function; maximum real eigenvalue (star marker) on principal branch ($b = 0$). **(B)** Linear interpolation between rate ($\alpha = 0$) and spiking model ($\alpha = 1$) by numerical integration of Senk et al. (2020, Eq. 30) (solid line) and by numerically solving the characteristic equation in Senk et al. (2020, Eq. 29) (circular markers). Star markers at same data points as in panel (A). Similar displays as in Senk et al. (2020, Figure 6).

```
nnmt.lif.exp.transfer_function(space_model)
nnmt.lif.exp.fit_transfer_function(space_model)
```

Figure 8B illustrates the amplitude and phase of the transfer function and its fit for a few $(\mu,\, \sigma)$ combinations. The plots of **Figure 8C** show the fitted time constants, the fitted excitatory weight, and the combined fit error. The inhibitory weight is proportional to the excitatory one in the same way as the post-synaptic current amplitudes.

3.4.3 Linear stability analysis of spatially structured model with delay

The rate model with fitted parameters combined with the spatial connectivity profile is considered a neural field model, which lends itself to analytical linear stability analysis, as described in detail in Senk et al. (2020, A. Linear stability analysis of a neural-field model). In brief, this analysis seeks for pattern-forming instabilities or, in other words, for transitions from homogeneous steady states to oscillatory states as a function of a bifurcation parameter, here the delay d. The complex-valued, temporal eigenvalue λ of the linearized time-delay system is an indicator for the system's overall stability and can serve as a predictor for temporal oscillations, whereas the spatial oscillations are characterized by the real-valued wave number k. Solutions that relate λ and k with the model parameters are given by a characteristic equation, which in our case reads (Senk et al., 2020, Eq. 7):

$$\lambda_b(k) = -\frac{1}{\tau} + \frac{1}{d} W_b\left(c\,(k)\,\frac{d}{\tau} \mathrm{e}^{\frac{d}{\tau}}\right) \quad , \tag{13}$$

with the time constant of the rate model τ, the multi-valued Lambert W_b function on branch b (Corless et al., 1996), and the effective connectivity profile $c\,(k)$, which combines the weights w and the Fourier transforms of the boxcar-shaped spatial connectivity profiles. NNMT provides an implementation to solve this characteristic equation in its `linear_stability` module using the `spatial` module for the profile:

```
connectivity = W_rate * nnmt.spatial._spatial_profile_boxcar(
    k_wavenumber, space_model.network_params['width'])
eigenvalue = nnmt.linear_stability._solve_characteristic_equation_lambertw(
    branch_nr, tau_rate, space_model.network_params['delay'], connectivity)
```

Figure 9A shows that the computed eigenvalues come for the given network parameters in complex conjugate pairs. Temporal oscillations are expected to occur if the real part of the eigenvalue on the largest branch becomes positive; the oscillation frequency can then be read off the imaginary part of that eigenvalue. In this example, the largest eigenvalue λ^* on the principle branch ($b = 0$) has a real part that is just above zero. The respective wave number k^* is positive, which indicates spatial oscillations as well. The oscillations in both time and space predicted for the rate model imply that the activity of the corresponding spiking model might exhibit wave trains, i.e., temporally and spatially periodic patterns. The predicted propagation speed of the wave trains is given by the phase velocity $\mathrm{Im}\left[\lambda^*\right]/k^*$.

To determine whether the results obtained with the rate model are transferable to the spiking model, **Figure 9B** interpolates the analytical solutions of the rate model ($\alpha = 0$) to solutions of the spiking model ($\alpha = 1$), which can only be computed numerically; for details see Senk et al. (2020, F.2 Linear interpolation between the transfer functions). Since the eigenvalues estimated this way show only little differences between rate and spiking model, we conclude that predictions from the rate model should hold also for the spiking model in the parameter regime tested. Following the argument of Senk et al. (2020), the predicted pattern formation could next be tested in a numerical simulation of the discrete spiking network model (for such results with the parameters used here, see their Figure 10(c) for $d = 1.5\,\mathrm{ms}$).

4 DISCUSSION

Mean-field theory grants important insights into the dynamics of neuronal networks. However, the lack of a publicly available numerical implementation for most methods entails a significant initial investment of time and effort prior to any scientific investigations, thereby slowing down progress.

This paper describes the open-source toolbox NNMT that provides access to various neuronal network model analyses based on mean-field theory. As use cases, we reproduce known results from the literature: The non-linear relation between the firing rates and the external input of an E-I-network (Sanzeni et al., 2020). The firing rates of a cortical microcircuit model, its response to oscillatory input, its power spectrum, and the identification of the connections that predominantly contribute to the model's high frequency oscillations (Bos et al., 2016; Schuecker et al., 2015). Finally, pattern formation in a spiking network, analyzed by mapping it to a rate model and conducting a linear stability analysis (Senk et al., 2020).

Here, we start with a general discussion on the benefits and drawbacks of collecting methods in a common toolbox. We expand on further use cases for NNMT and also point out the inherent limitations of analytical methods for neuronal network analysis. We conclude with suggestions on how NNMT might overcome some of these limitations and how the toolbox may grow and develop in the future.

4.1 Using a Toolbox vs. Writing Your Own Code

Why is there not yet a toolbox like this (of which we are aware of)? And consequently, is there actually a need for a toolbox like this? Here, we would like to share our thoughts on these questions (see also Riquelme and Gjorgjieva, 2021).

Without a toolbox, everybody who wants to use an analytical method for network model analysis, needs to implement them on their own. This has an advantage: a detailed understanding of the theory behind the method and the implementation itself. Accordingly, a scientist following this path will know exactly how to apply the method and, most importantly, when the method cannot be applied and conversely when it will lead to misleading or wrong results. Providing those tools in a ready-to-use fashion comes with the danger of enticing users into applying those tools without understanding their limitations. And this might lead them to draw false conclusions based on the results they get from using the toolbox. This can only be avoided by making the users aware of the limitations a method comes with and providing sufficient information in an easy-to-find and comprehensible form. Therefore, in NNMT, we try to address this issue by raising warnings whenever the valid parameter regimes are left and referring the user to the package's documentation, where we try our best to make all the necessary information available. Of course, the best protection against running into those problems in the first place is reading the documentation before using the tools.

However, implementing everything on your own also has disadvantages: It takes time and therefore money, which possibly is invested into the same tool over and over again by different groups around the world; which also can be seen as a waste of resources. Often, implementing and debugging analytical tools is not straightforward and can be a matter of several weeks. This naturally implicates a certain complexity barrier, beyond which such tools, which might be dependent on other tools, cannot develop, as the effort of implementing them becomes too large. In a similar vein, which kind of questions researchers pursue is limited by and therefore depends on the tools they have at hand. The availability of sophisticated neural network simulators for example has lead to the development of conceptually new and more complex neural network models, precisely because their users did not have to bother about implementation details any longer and could instead focus on actual research questions. We hope that collecting useful implementations of analytical tools for network model analysis will have a similar effect on the development of new tools and that it might lead to new, creative ways of applying them.

Finally, having an open-source toolbox, everybody can contribute to, has the compelling advantage of leveraging the knowledge of a large part of the scientific community. Such a toolbox can serve as a platform for collecting standard implementations of methods, which have been tested thoroughly and have been validated by many users. Of course, if a toolbox does not have the functionality a user might need, it needs some extra effort to first learn how the toolbox works and to find out where necessary changes need to be made. But instead of coding everything from scratch, it might only be a few lines that need to be modified. And most importantly, in the end not only the own lab but the whole community may benefit from this extra effort.

4.2 Use Cases

In Section 3, we present concrete examples of how to apply some of the tools available. Here, we discuss further, more general use cases for NNMT: parameter space exploration and fast prototyping, developing an intuitive understanding of network models, extracting network mechanisms, investigating hypotheses using limited computational resources, and hybrid approaches.

Analytical methods have the advantage of being fast, and typically they only require a limited amount of computational resources. All tools currently implemented in NNMT can be run on a notebook without any problems. And the computational costs for calculating analytical estimates of dynamical network properties like firing rates, as opposed to the costs of running simulations of a network model, are independent of the number of neurons the network is composed of. This becomes especially relevant if one wants to do parameter space explorations, for which many simulations have to be performed. To speed up prototyping, a modeler can first perform a parameter scan using analytical tools from NNMT to get an estimate of the right parameter regimes and subsequently run simulations on this restricted set of parameters to arrive at the final model parameters.

But additionally to speeding up undirected parameter space explorations, analytical methods might guide parameter space explorations in another way: namely, by giving some intuitive understanding of the relation between network model parameters and network dynamics. The prime example implemented in NNMT is the sensitivity measure presented in Section 3.3.3, which provides a "frequency-dependent connectivity map that reveals connections crucial for the peak amplitude and frequency of the observed oscillations [of a network model] and identifies the minimal circuit generating a given frequency" (Bos et al., 2016). This illustrates a case in which a mean-field method can inform the modeler about the origin of a model's dynamics, and it identifies which model parameters need to be adjusted to modify that dynamics.

Similarly, a modeler might be interested in extracting network mechanisms, answering the question, which features of a network cause some dynamics to happen. We have included a method for mapping the parameters of a spiking network model to a rate network model, which is presented in Section 3.4. Using this method, one can check whether the dynamical properties under investigation are still present in a simulation of a simpler rate network. Consequently, one can conclude whether spiking dynamics is a crucial component for the occurrence of the respective dynamical property, or whether it is sufficient to study the simpler model. This way, NNMT can help identifying the essential components that lead to some dynamical features.

Another way to take advantage of analytical tools, is to investigate hypotheses directly, employing those methods. In Section 3.2.1 the toolbox is used to show non-trivial effects regarding the firing rates. Users could first perform a similar analysis for their research question, using the computationally cheap mean-field methods, thereby deepening their understanding of the problem, and then apply for compute time on a high performance cluster afterwards to check those predictions in simulations. Thus, NNMT can serve as an additional instrument for investigating research questions, especially for researchers who do not have free access to a high performance computer. Of course, the limitations explained in Section 4.3 always have to be taken into account to avoid misleading conclusions.

Finally, NNMT can be used for hybrid approaches, where one quantity is extracted from simulations or experiments, and the toolbox is then used to calculate another quantity that might not be directly accessible otherwise. Getting access to possibly abstract, but informative measures opens up the possibility to conduct new kinds of experiments and simulations, which investigate the behavior of such measures under possibly complex interventions that could not be done using analytical tools alone.

4.3 Limitations

NNMT has two major limitations: restrictions of valid parameter regimes due to necessary approximations in the analytics, and the restriction to network, neuron, and synapse models, as well as observables, for which a mean-field theory exists.

Analytical methods can provide good estimates of properties of a network model. But it is important to state that those methods almost always depend on some kinds of approximations, which can only be justified if the network under consideration fulfills certain assumptions. If those assumptions are not met, at least approximately, one cannot assure that the corresponding analytical methods give valid results. For example, NNMT calculates the firing rates for leaky integrate-and-fire neurons with exponential synapses using the results from Fourcaud and Brunel (2002). This requires the neuronal populations to be large enough, it requires them to receive a high enough number of uncorrelated inputs, such that the central limit theorem can be applied, and the solution depends on the assumption that the synaptic time constant is small compared to the membrane time constant. Fortunately, those assumptions are known and some of them can be checked for. However, there are assumptions that cannot be tested, as they would necessitate a theory going beyond the mean-field approximation, such as the network state being sufficiently uncorrelated. If a tool from NNMT is applied outside of its valid parameter ranges, NNMT will raise a warning, informing the user about the issue, whenever this is possible.

To better understand the sources of inaccuracies in NNMT, it is helpful to consider a comparison to direct simulations of network models. Simulations only suffer from inaccuracies due to the numerical implementation, which can be improved upon by using enhanced numerical schemes or simply choosing smaller integration steps. Numerical inaccuracies occur in NNMT as well, and they can be remedied in a similar fashion. We address this problem with an extensive test suite based on previously published results. But often, the assumptions discussed in the previous paragraph cannot be met perfectly. For example some analytical estimates might only be exact if the number of neurons in the network is infinite. Obviously this cannot be achieved in a realistic network. Therefore, the approximations applied in analytical methods can introduce a second source of inaccuracy, which originates from neglecting higher order contributions. This kind of inaccuracy cannot be resolved by improving some hyperparameters, as it depends directly on the choice of network parameters. This explains the discrepancy between networks model simulations and analytical results. Nevertheless, analytical methods will lead to reliable estimates if applied within the valid parameter regimes (for example see **Figure 3B**).

Finally, mean-field methods will always be restricted to network, neuron, and synapse models for which the relation between model parameters and activity statistics can be derived. If the model under consideration does not allow deriving such a relation, resorting to a similar model, as done in Section 3.2.2, may be a viable solution. But if a model becomes too complex, it is usually impossible to derive any analytical results. In such cases, one might prefer simulations (Einevoll et al., 2019). However, analytical methods come with advantages that no other approach can offer, as we have shown in Section 4.2. In the following section we explain how interested colleagues can support the community in fully utilizing these advantages.

4.4 How to Contribute and Outlook

As discussed in Section 2, NNMT has been designed for modularity, flexibility, and extensibility. The aim is to provide a platform which can fit many different mean-field methods. NNMT is an open-source toolbox, and we would like to encourage scientists to contribute by improving existing methods and sharing their own methods and implementations via the toolbox. This can be done via the standard pull request workflow on GitHub. If you are interested and would like to know more about the procedure, we refer you to the "Contributors guide" of the official documentation of NNMT[4].

[4] https://nnmt.readthedocs.io/

A toolbox like NNMT always is an ongoing project, and there are various aspects that can be improved. Here we shortly discuss three of them: parallelization, extension of valid parameter regimes, and different neuron types and network architectures.

NNMT in its current state is partly vectorized but it currently does not include parallelized methods, e.g., using multiprocessing or MPI for Python (`mpi4py`). Vectorization relies on `NumPy` (Harris et al., 2020) and `SciPy` (Virtanen et al., 2020) which are thread-parallel for specific backends, e.g., IntelMKL. With the tools available in the toolbox at the moment, run-time only becomes an issue in extensive parameter scans, for instance, when the transfer function needs to be calculated for a large range of frequencies. To further reduce the runtime, the code could be made fully vectorized. Alternatively, parallelization of many tools in NNMT is straightforward, as many of them include `for` loops over the different populations of a network model and `for` loops over the different analysis frequencies.

As discussed in Section 4.3, many mean-field methods only give valid results for certain parameter ranges. But often, there exist different approximations for the same quantity, valid in complementary parameter regimes. A concrete example is the currently implemented version of the transfer function for leaky integrate-and-fire neurons, based on the work of Schuecker et al. (2015), which gives a good estimate for small synaptic time constants compared to the membrane time constant $\tau_\mathrm{s}/\tau_\mathrm{m} \ll 1$. A complementary estimate for long synaptic time constants $\tau_\mathrm{m}/\tau_\mathrm{s} \ll 1$, which could be implemented, has been developed by Moreno-Bote and Parga (2006). Similarly, the current implementation of the firing rates of leaky integrate-and-fire neurons is based on the work of Fourcaud and Brunel (2002), and only valid for $\tau_\mathrm{s}/\tau_\mathrm{m} \ll 1$. But van Vreeswijk and Farkhooi (2019) have developed a method accurate for all combinations of synaptic and membrane time constants. Adding such complementary and improved tools will enhance the applicability of the toolbox.

Finally, due to the research focus at our lab, NNMT presently mainly contains tools for LIF neurons. But the structure of NNMT allows for adding methods for different neuron types, like binary (Ginzburg and Sompolinsky, 1994) or conductance-based neurons (Izhikevich, 2007; Richardson, 2007). In a similar manner, most methods focus on networks with random connectivity, but we already added some tools for networks with spatially organized connectivity. We hope that in the future many scientists will contribute to this collection of analytical methods for neuronal network model analysis, such that at some point, we will have tools from all parts of mean-field theory of neuronal networks, made accessible in a usable format to all neuroscientists.

AUTHOR CONTRIBUTIONS

HB and MH developed and implemented the code base and the inital version of the toolbox. ML, JS, and SE designed the current version of the toolbox. ML implemented the current version of the toolbox, vectorized and generalized tools, developed and implemented the test suite, wrote the documentation, and created the example shown in Section 3.2.2. AvM improved the numerics of the firing rate integration (Appendix Section 5.1) and created the example shown in Section 3.2.1. SE implemented integration tests, improved the functions related to the `sensitivity_measure`, and created the examples shown in Section 3.3. JS developed and implemented the tools used in Section 3.4 and created the respective example. ML, JS, SE, AvM, and MH wrote this article. All authors approved the submitted version.

ACKNOWLEDGMENTS

We would like to thank Jannis Schuecker, who has contributed to the development and implementation of the code base and the inital version of the toolbox. This project has received funding from the European Union's

Horizon 2020 Framework Programme for Research and Innovation under Specific Grant Agreement No. 720270 (HBP SGA1), 785907 (HBP SGA2), and 945539 (HBP SGA3), and has been partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 368482240/GRK2416. This research was supported by the Joint Lab "Supercomputing and Modeling for the Human Brain".

5 APPENDIX

5.1 Siegert Implementation

Here, we describe how we solve the integral in Eq. (2) numerically in a fully vectorized manner. The difficulty in Eq. (2), $\phi(\mu,\sigma) = 1/[\tau_\mathrm{r} + \tau_\mathrm{m}\sqrt{\pi} I(\widetilde{V}_0, \widetilde{V}_\mathrm{th})]$ where $\widetilde{V}_0 = \widetilde{V}_0(\mu,\sigma)$ and $\widetilde{V}_\mathrm{th} = \widetilde{V}_\mathrm{th}(\mu,\sigma)$ are determined by either Eq. (3) or Eq. (7), is posed by the integral

$$I(\widetilde{V}_0, \widetilde{V}_\mathrm{th}) = \int_{\widetilde{V}_0}^{\widetilde{V}_\mathrm{th}} e^{s^2}(1+\mathrm{erf}(s))\mathrm{d}s \quad . \tag{14}$$

This integral is problematic due to the multiplication of e^{s^2} and $1+\mathrm{erf}(s)$ in the integrand which leads to overflow and loss of significance.

To address this, we split the integral into different domains depending on the sign of the integration variable. Furthermore, we use the scaled complementary error function

$$\mathrm{erf}(s) = 1 - e^{-s^2}\mathrm{erfcx}(s) \tag{15}$$

to extract the leading exponential contribution. Importantly, $\mathrm{erfcx}(s)$ decreases monotonically from $\mathrm{erfcx}(0) = 1$ with power law asymptotics $\mathrm{erfcx}(s) \sim 1/(\sqrt{\pi}s)$, hence it does not contain any exponential contribution. For positive s, the exponential contribution in the prefactor of $\mathrm{erfcx}(s)$ cancels the e^{s^2} factor in the integrand. For negative s, the integrand simplifies even further to $e^{s^2}(1+\mathrm{erf}(-s)) = \mathrm{erfcx}(s)$ using $\mathrm{erf}(-s) = -\mathrm{erf}(s)$. In addition to $\mathrm{erfcx}(s)$, we employ the Dawson function

$$D(s) = e^{-s^2}\int_0^s e^{r^2}\mathrm{d}r \tag{16}$$

to solve some of the integrals analytically. The Dawson function has a power law tail, $D(s) \sim 1/(2s)$; hence, it also does not carry an exponential contribution. Both $\mathrm{erfcx}(s)$ and the Dawson function are fully vectorized in `SciPy` (Virtanen et al., 2020).

Any remaining integrals are solved using Gauss–Legendre quadrature (Press et al., 2007). By construction, Gauss–Legendre quadrature of order k solves integrals of polynomials up to degree k on the interval $[-1,1]$ exactly. Thus, it gives very good results if the integrand is well approximated by a polynomial of degree k. The quadrature rule itself is

$$\int_a^b f(s)\mathrm{d}s \approx \frac{b-a}{2}\sum_{i=1}^{k} w_i f\left(\frac{b-a}{2}u_i + \frac{b+a}{2}\right) \quad , \tag{17}$$

where the u_i are the roots of the Legendre polynomial of order k and the w_i are appropriate weights such that a polynomial of degree k is integrated exactly. We use a fixed order quadrature for which Eq. (17) is straightforward to vectorize to multiple a and b. We determine the order of the quadrature iteratively by

comparison with an adaptive quadrature rule; usually, a small order $k = O(10)$ already yields very good results for an $\mathrm{erfcx}(s)$ integrand.

Inhibitory Regime

First, we consider the case where lower and upper bound of the integral are positive, $0 < \widetilde{V}_0 < \widetilde{V}_\mathrm{th}$. This corresponds to strongly inhibitory mean input. Expressing the integrand in terms of $\mathrm{erfcx}(s)$ and using the Dawson function, we get

$$I(\widetilde{V}_0, \widetilde{V}_\mathrm{th}) = 2e^{\widetilde{V}_\mathrm{th}^2} D(\widetilde{V}_\mathrm{th}) - 2e^{\widetilde{V}_0^2} D(\widetilde{V}_0) - \int_{\widetilde{V}_0}^{\widetilde{V}_\mathrm{th}} \mathrm{erfcx}(s)\mathrm{d}s \quad .$$

The remaining integral is evaluated using Gauss–Legendre quadrature, Eq. (17). We extract the leading contribution $e^{\widetilde{V}_\mathrm{th}^2}$ from the denominator in Eq. (2) and arrive at

$$\phi(\mu, \sigma) = \frac{e^{-\widetilde{V}_\mathrm{th}^2}}{\tau_\mathrm{r} e^{-\widetilde{V}_\mathrm{th}^2} + \tau_\mathrm{m}\sqrt{\pi}\left(2D(\widetilde{V}_\mathrm{th}) - 2e^{-\widetilde{V}_\mathrm{th}^2+\widetilde{V}_0^2} D(\widetilde{V}_0) - e^{-\widetilde{V}_\mathrm{th}^2} \int_{\widetilde{V}_0}^{\widetilde{V}_\mathrm{th}} \mathrm{erfcx}(s)\mathrm{d}s\right)} \quad . \tag{18}$$

Extracting $e^{\widetilde{V}_\mathrm{th}^2}$ from the denominator reduces the latter to $2\tau_\mathrm{m}\sqrt{\pi}D(\widetilde{V}_\mathrm{th})$ and exponentially small correction terms (remember $0 < \widetilde{V}_0 < \widetilde{V}_\mathrm{th}$ because $V_0 < V_\mathrm{th}$), thereby preventing overflow.

Excitatory Regime

Second, we consider the case where lower and upper bound of the integral are negative, $\widetilde{V}_0 < \widetilde{V}_\mathrm{th} < 0$. This corresponds to strongly excitatory mean input. In this regime, we change variables $s \to -s$ to make the domain of integration positive. Using $\mathrm{erf}(-s) = -\mathrm{erf}(s)$ as well as $\mathrm{erfcx}(s)$, we get

$$I(\widetilde{V}_0, \widetilde{V}_\mathrm{th}) = \int_{|\widetilde{V}_\mathrm{th}|}^{|\widetilde{V}_0|} \mathrm{erfcx}(s)\mathrm{d}s \quad .$$

Thus, we evaluate Eq. (2) as

$$\phi(\mu, \sigma) = \frac{1}{\tau_\mathrm{r} + \tau_\mathrm{m}\sqrt{\pi} \int_{|\widetilde{V}_\mathrm{th}|}^{|\widetilde{V}_0|} \mathrm{erfcx}(s)\mathrm{d}s} \quad . \tag{19}$$

In particular, there is no exponential contribution involved in this regime.

Intermediate Regime

Last, we consider the remaining case $\widetilde{V}_0 \leq 0 \leq \widetilde{V}_\mathrm{th}$. We split the integral at zero and use the previous steps for the respective parts to get

$$I(\widetilde{V}_0, \widetilde{V}_\mathrm{th}) = 2e^{\widetilde{V}_\mathrm{th}^2} D(\widetilde{V}_\mathrm{th}) + \int_{\widetilde{V}_\mathrm{th}}^{|\widetilde{V}_0|} \mathrm{erfcx}(s)\mathrm{d}s \quad .$$

Note that the sign of the second integral depends on whether $|\widetilde{V}_0| > \widetilde{V}_{\mathrm{th}}$ $(+)$ or not $(-)$. Again, we extract the leading contribution $e^{\widetilde{V}_{\mathrm{th}}^2}$ from the denominator in Eq. (2) and arrive at

$$\phi(\mu,\sigma) = \frac{e^{-\widetilde{V}_{\mathrm{th}}^2}}{\tau_{\mathrm{r}} e^{-\widetilde{V}_{\mathrm{th}}^2} + \tau_{\mathrm{m}}\sqrt{\pi}\left(2D(\widetilde{V}_{\mathrm{th}}) + e^{-\widetilde{V}_{\mathrm{th}}^2}\int_{\widetilde{V}_{\mathrm{th}}}^{|\widetilde{V}_0|}\mathrm{erfcx}(s)\mathrm{d}s\right)} \quad . \tag{20}$$

As before, extracting $e^{\widetilde{V}_{\mathrm{th}}^2}$ from the denominator prevents overflow.

Deterministic Limit

The deterministic limit $\sigma \to 0$ corresponds to $|\widetilde{V}_0|, |\widetilde{V}_{\mathrm{th}}| \to \infty$ for both Eq. (3) and Eq. (7). In the inhibitory and the intermediate regime, we see immediately that $\phi(\mu, \sigma \to 0) \to 0$ due to the dominant contribution $e^{-\widetilde{V}_{\mathrm{th}}^2}$. In the excitatory regime, we use the asymptotics $\mathrm{erfcx}(s) \sim 1/(\sqrt{\pi}s)$ to get

$$I(\widetilde{V}_0, \widetilde{V}_{\mathrm{th}}) \to \int_{|\widetilde{V}_{\mathrm{th}}|}^{|\widetilde{V}_0|} \frac{1}{\sqrt{\pi}s}\mathrm{d}s = \frac{1}{\sqrt{\pi}} \ln \frac{|\widetilde{V}_0|}{|\widetilde{V}_{\mathrm{th}}|} \quad .$$

Inserting this into Eq. (2) yields

$$\phi(\mu,\sigma) \to \begin{cases} \frac{1}{\tau_{\mathrm{r}} + \tau_{\mathrm{m}} \ln \frac{\mu - V_0}{\mu - V_{\mathrm{th}}}} & \text{if } \mu > V_{\mathrm{th}} \\ 0 & \text{otherwise} \end{cases} , \tag{21}$$

which is the firing rate of a leaky integrate-and-fire neuron driven by a constant input (Gerstner et al., 2014). Thus, this implementation also tolerates the deterministic limit of a very small noise intensity σ.

5.2 Microcircuit Parameters

Symbol	**Value** Potjans and Diesmann (2014)	**Value** Bos et al. (2016)	**Description**
$K_{\mathrm{4E,4I}}$	795	675	In-degree from 4I to 4E
$K_{\mathrm{4E,ext}}$	2100	1780	External in-degree to 4E
$D(\omega)$	none	truncated Gaussian	Delay distribution
$d_e \pm \delta d_e$	1.5 ± 0.75 ms	1.5 ± 1.5 ms	Mean and standard deviation of excitatory delay
$d_i \pm \delta d_i$	0.75 ± 0.375 ms	0.75 ± 0.75 ms	Mean and standard deviation of inhibitory delay

Table 1. Parameter adaptions used here are introduced by Bos et al. (2016) compared to original microcircuit model. K_{ij} denotes the in-degrees from population j to population i. The delays in the simulated networks were drawn from a truncated Gaussian distribution with the given mean and standard deviation. The mean-field approximation of the microcircuit by Potjans and Diesmann (2014) assumes the delay to be fixed at the mean value, which is specified in the toolbox by setting the parameter `delay_dist` to `none`.

REFERENCES

Abramowitz, M. and Stegun, I. A. (1974). *Handbook of Mathematical Functions: with Formulas, Graphs, and Mathematical Tables* (New York: Dover Publications). doi:10.1119/1.15378

Amari, S.-I. (1975). Homogeneous nets of neuron-like elements. *Biol. Cybern.* 17, 211–220. doi:10.1007/BF00339367

Amari, S.-I. (1977). Dynamics of pattern formation in lateral-inhibition type neural fields. *Biol. Cybern.* 27, 77–87. doi:10.1007/bf00337259

Amit, D. J. and Brunel, N. (1997a). Dynamics of a recurrent network of spiking neurons before and following learning. *Network: Comp. Neural Sys.* 8, 373–404. doi:10.1088/0954-898x_8_4_003

Amit, D. J. and Brunel, N. (1997b). Model of global spontaneous activity and local structured activity during delay periods in the cerebral cortex. *Cereb. Cortex* 7, 237–252. doi:10.1093/cercor/7.3.237

Bos, H., Diesmann, M., and Helias, M. (2016). Identifying anatomical origins of coexisting oscillations in the cortical microcircuit. *PLOS Comput. Biol.* 12, e1005132. doi:10.1371/journal.pcbi.1005132

Braitenberg, V. and Schüz, A. (1998). *Cortex: Statistics and Geometry of Neuronal Connectivity* (Berlin: Springer-Verlag), 2nd edn.

Bressloff, P. C. (2012). Spatiotemporal dynamics of continuum neural fields. *Journal of Physics A: Mathematical and Theoretical* 45, 033001. doi:10.1088/1751-8113/45/3/033001

Bressloff, P. C., Cowan, J. D., Golubitsky, M., Thomas, P. J., and Wiener, M. C. (2001). Geometric visual hallucinations, euclidean symmetry and the functional architecture of striate cortex. *Phil. Trans. R. Soc. B* 356, 299–330. doi:10.1098/rstb.2000.0769

Brunel, N. (2000). Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons. *Journal of Computational Neuroscience* 8, 183–208. doi:10.1023/a:1008925309027

Brunel, N., Chance, F. S., Fourcaud, N., and Abbott, L. F. (2001). Effects of synaptic noise and filtering on the frequency response of spiking neurons. *Phys. Rev. Lett.* 86, 2186–2189. doi:10.1103/physrevlett.86.2186

Brunel, N. and Hakim, V. (1999). Fast global oscillations in networks of integrate-and-fire neurons with low firing rates. *Neural Comput.* 11, 1621–1671. doi:10.1162/089976699300016179

Brunel, N. and Latham, P. (2003). Firing rate of the noisy quadratic integrate-and-fire neuron. *Neural Comput.* 15, 2281–2306

Buice, M. A. and Chow, C. C. (2013). Beyond mean field theory: statistical field theory for neural networks. *Journal of Statistical Mechanics: Theory and Experiment* 2013, P03003

Cain, N., Iyer, R., Koch, C., and Mihalas, S. (2016). The computational properties of a simplified cortical column model. *PLOS Comput. Biol.* 12, e1005045. doi:10.1371/journal.pcbi.1005045

Coombes, S. (2005). Waves, bumps, and patterns in neural field theories. *Biological Cybernetics* 93, 91–108. doi:10.1007/s00422-005-0574-y

Coombes, S., bei Graben, P., Potthast, R., and Wright, J. (2014). *Neural Fields. Theory and Applications* (Springer-Verlag Berlin Heidelberg)

Corless, R. M., Gonnet, G. H., Hare, D. E. G., Jeffrey, D. J., and Knuth, D. E. (1996). On the lambert w function. *Advances in Computational Mathematics* 5, 329–359. doi:10.1007/BF02124750

Dahmen, D., Layer, M., Deutz, L., Dąbrowska, P. A., Voges, N., von Papen, M., et al. (2021). Global organization of neuronal activity only requires unstructured local connectivity. *bioRxiv* , 2020–07

Dasbach, S., Tetzlaff, T., Diesmann, M., and Senk, J. (2021). Prominent characteristics of recurrent neuronal networks are robust against low synaptic weight resolution. *arXiv* , 2105.05002 [q–bio.NC]

DeFelipe, J., Alonso-Nanclares, L., and Arellano, J. (2002). Microstructure of the neocortex: comparative aspects. *J. Neurocytol.* 31, 299–316. doi:10.1023/A:1024130211265

Einevoll, G. T., Destexhe, A., Diesmann, M., Grün, S., Jirsa, V., de Kamps, M., et al. (2019). The Scientific Case for Brain Simulations. *Neuron* 102, 735–744. doi:10.1016/j.neuron.2019.03.027

Ermentrout, G. B. and Cowan, J. D. (1979). A mathematical theory of visual hallucination patterns. *Biol. Cybern.* 34, 137–150

Fourcaud, N. and Brunel, N. (2002). Dynamics of the firing probability of noisy integrate-and-fire neurons. *Neural Comput.* 14, 2057–2110

Fourcaud-Trocmé, N., Hansel, D., van Vreeswijk, C., and Brunel, N. (2003). How spike generation mechanisms determine the neuronal response to fluctuating inputs. *Journal of Neuroscience* 23, 11628–11640

Gerstner, W., Kistler, W. M., Naud, R., and Paninski, L. (2014). *Neuronal Dynamics. From Single Neurons to Networks and Models of Cognition* (Cambridge: Cambridge University Press)

Giese, M. A. (2012). *Dynamic neural field theory for motion perception*, vol. 469 (Springer Science & Business Media)

Ginzburg, I. and Sompolinsky, H. (1994). Theory of correlations in stochastic neural networks. *Phys. Rev. E* 50, 3171–3191. doi:10.1103/PhysRevE.50.3171

Goldenfeld, N. (1992). *Lectures on phase transitions and the renormalization group* (Reading, Massachusetts: Perseus books)

Golosio, B., Tiddia, G., Luca, C. D., Pastorelli, E., Simula, F., and Paolucci, P. S. (2021). Fast simulations of highly-connected spiking cortical models using GPUs. *Front. Comput. Neurosci.* 15. doi:10.3389/fncom.2021.627620

Grabska-Barwinska, A. and Latham, P. (2014). How well do mean field theories of spiking quadratic-integrate-and-fire networks work in realistic parameter regimes? *Journal of Computational Neuroscience* 36, 469–81

Grytskyy, D., Tetzlaff, T., Diesmann, M., and Helias, M. (2013). A unified view on weakly correlated recurrent networks. *Front. Comput. Neurosci.* 7, 131. doi:10.3389/fncom.2013.00131

Hagen, E., Dahmen, D., Stavrinou, M. L., Lindén, H., Tetzlaff, T., van Albada, S. J., et al. (2016). Hybrid scheme for modeling local field potentials from point-neuron networks. *Cereb. Cortex* 26, 4461–4496. doi:10.1093/cercor/bhw237

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., et al. (2020). Array programming with NumPy. *Nature* 585, 357–362. doi:10.1038/s41586-020-2649-2

Helias, M., Tetzlaff, T., and Diesmann, M. (2014). The correlation structure of local cortical networks intrinsically results from recurrent dynamics. *PLOS Comput. Biol.* 10, e1003428. doi:10.1371/journal.pcbi.1003428

Hertz, J. (2010). Cross-correlations in high-conductance states of a model cortical network. *Neural Comput.* 22, 427–447

Izhikevich, E. M. (2007). *Dynamical Systems in Neuroscience: The Geometry of Excitability and Bursting* (MIT Press)

Jirsa, V. K. and Haken, H. (1996). Field theory of electromagnetic brain activity. *Phys. Rev. Lett.* 77, 960

Jirsa, V. K. and Haken, H. (1997). A derivation of a macroscopic field theory of the brain from the quasi-microscopic neural dynamics. *Physica D: Nonlinear Phenomena* 99, 503–526

Knight, J. C. and Nowotny, T. (2018). GPUs outperform current HPC and neuromorphic solutions in terms of speed and energy when simulating a highly-connected cortical model. *Front. Neurosci.* 12, 1–19. doi:10.3389/fnins.2018.00941

Laing, C. R. and Troy, W. C. (2003). Two-bump solutions of amari-type models of neuronal pattern formation. *Physica D: Nonlinear Phenomena* 178, 190–218

Laing, C. R., Troy, W. C., Gutkin, B., and Ermentrout, B. G. (2002). Multiple bumps in a neuronal model of working memory. *SIAM J. Appl. Math.* 63, 62–97. doi:10.1137/s0036139901389495

[Dataset] Layer, M., Senk, J., Essink, S., van Meegen, A., Bos, H., and Helias, M. (2021). NNMT 1.0.0. doi:10.5281/zenodo.5779549

Lindner, B. (2004). Interspike interval statistics of neurons driven by colored noise. *Phys. Rev. E* 69, 0229011–0229014

Lindner, B., Doiron, B., and Longtin, A. (2005). Theory of oscillatory firing induced by spatially correlated noise and delayed inhibitory feedback. *Phys. Rev. E* 72, 061919. doi:10.1103/physreve.72.061919

Lindner, B. and Longtin, A. (2005). Effect of an exponentially decaying threshold on the firing statistis of a stochastic integate-and-fire neuron. *Journal of Theoretical Biology* 232, 505–521

Lindner, B. and Schimansky-Geier, L. (2001). Transmission of noise coded versus additive signals through a neuronal ensemble. *Phys. Rev. Lett.* 86, 2934–2937. doi:10.1103/physrevlett.86.2934

Mattia, M., Biggio, M., Galluzzi, A., and Storace, M. (2019). Dimensional reduction in networks of non-markovian spiking neurons: Equivalence of synaptic filtering and heterogeneous propagation delays. *PLOS Comput. Biol.* 15, e1007404. doi:10.1371/journal.pcbi.1007404

Montbrió, E., Pazó, D., and Roxin, A. (2015). Macroscopic description for networks of spiking neurons. *Phys Rev X* 5, 021028. doi:10.1103/PhysRevX.5.021028

Moreno-Bote, R. and Parga, N. (2006). Auto- and crosscorrelograms for the spike response of leaky integrate-and-fire neurons with slow synapses. *Phys. Rev. Lett.* 96, 028101

Nunez, P. L. (1974). The brain wave equation: a model for the eeg. *Mathematical Biosciences* 21, 279–297

[Dataset] Olver, F. W. J., Olde Daalhuis, A. B., Lozier, D. W., Schneider, B. I., Boisvert, R. F., Clark, C. W., et al. (2021). NIST Digital Library of Mathematical Functions. http://dlmf.nist.gov/, Release 1.1.2 of 2021-06-15

Ostojic, S. (2014). Two types of asynchronous activity in networks of excitatory and inhibitory spiking neurons. *Nat. Neurosci.* 17, 594–600. doi:10.1038/nn.3658

Ostojic, S. and Brunel, N. (2011). From spiking neuron models to linear-nonlinear models. *PLOS Comput. Biol.* 7, e1001056

Potjans, T. C. and Diesmann, M. (2014). The cell-type specific cortical microcircuit: Relating structure and activity in a full-scale spiking network model. *Cereb. Cortex* 24, 785–806. doi:10.1093/cercor/bhs358

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (2007). *Numerical Recipes: The Art of Scientific Computing* (Cambridge University Press), 3rd edn.

Renart, A., De La Rocha, J., Bartho, P., Hollender, L., Parga, N., Reyes, A., et al. (2010). The asynchronous state in cortical circuits. *Science* 327, 587–590. doi:10.1126/science.1179850

Richardson, M. J. E. (2007). Firing-rate response of linear and nonlinear integrate-and-fire neurons to modulated current-based and conductance-based synaptic drive. *Phys. Rev. E* 76, 1–15

Richardson, M. J. E. (2008). Spike-train spectra and network response functions for non-linear integrate-and-fire neurons. *Biol. Cybern.* 99, 381–392

Riquelme, J. L. and Gjorgjieva, J. (2021). Towards readable code in neuroscience. *Nat. Rev. Neurosci.* 22, 257–258

Rosenbaum, R. and Doiron, B. (2014). Balanced networks of spiking neurons with spatially dependent recurrent connections 4, 021039. *Phys. Rev. X* 4, 021039. doi:10.1103/PhysRevX.4.021039

Rosenbaum, R., Smith, M. A., Kohn, A., Rubin, J. E., and Doiron, B. (2017). The spatial structure of correlated neuronal variability. *Nat. Neurosci.* 20, 107–114. doi:10.1038/nn.4433

Sanzeni, A., Histed, M. H., and Brunel, N. (2020). Response nonlinearities in networks of spiking neurons. *PLOS Comput. Biol.* 16, e1008165

Schmidt, M., Bakker, R., Hilgetag, C. C., Diesmann, M., and van Albada, S. J. (2018). Multi-scale account of the network structure of macaque visual cortex. *Brain Struct. Func.* 223, 1409–1435. doi:10.1007/s00429-017-1554-4

Schöner, G. (2008). Dynamical systems approaches to cognition. *Cambridge handbook of computational cognitive modeling* , 101–126

Schuecker, J., Diesmann, M., and Helias, M. (2014). Reduction of colored noise in excitable systems to white noise and dynamic boundary conditions. *arXiv* , 1410.8799

Schuecker, J., Diesmann, M., and Helias, M. (2015). Modulated escape from a metastable state driven by colored noise. *Phys. Rev. E* 92, 052119. doi:10.1103/PhysRevE.92.052119

Schuecker, J., Goedeke, S., and Helias, M. (2018). Optimal sequence memory in driven random networks 8, 041029. *Phys. Rev. X* 8, 041029. doi:10.1103/PhysRevX.8.041029

Schwalger, T., Deger, M., and Gerstner, W. (2017). Towards a theory of cortical columns: From spiking neurons to interacting neural populations of finite size. *PLOS Comput. Biol.* 13, e1005507. doi:10.1371/journal.pcbi.1005507

Schwalger, T., Droste, F., and Lindner, B. (2015). Statistical structure of neural spiking under non-poissonian or other non-white stimulation. *Journal of computational neuroscience* 39, 29

Sejnowski, T. (1976). On the stochastic dynamics of neuronal interaction. *Biological cybernetics* 22, 203–211

Senk, J., Korvasová, K., Schuecker, J., Hagen, E., Tetzlaff, T., Diesmann, M., et al. (2020). Conditions for wave trains in spiking neural networks. *Phys. Rev. Res.* 2. doi:10.1103/physrevresearch.2.023174

Senk, J., Kriener, B., Djurfeldt, M., Voges, N., Jiang, H.-J., Schüttler, L., et al. (2021). Connectivity concepts in neuronal network modeling. *arXiv* , 2110.02883 [q–bio.NC]

Siegert, A. J. (1951). On the first passage time probability problem. *Physical Review* 81, 617–623

Sompolinsky, H., Crisanti, A., and Sommers, H. J. (1988). Chaos in random neural networks. *Phys. Rev. Lett.* 61, 259–262. doi:10.1103/PhysRevLett.61.259

Stiller, J. and Radons, G. (1998). Dynamics of nonlinear oscillators with random interactions. *Phys. Rev. E* 58, 1789

Tetzlaff, T., Helias, M., Einevoll, G. T., and Diesmann, M. (2012). Decorrelation of neural-network activity by inhibitory feedback. *PLOS Comput. Biol.* 8, e1002596. doi:10.1371/journal.pcbi.1002596

Toyoizumi, T. and Abbott, L. F. (2011). Beyond the edge of chaos: Amplification and temporal integration by recurrent networks in the chaotic regime. *Phys. Rev. E* 84, 051908

Trousdale, J., Hu, Y., Shea-Brown, E., and Josic, K. (2012). Impact of network structure and cellular response on spike time correlations. *PLOS Comput. Biol.* 8, e1002408

van Albada, S. J., Rowley, A. G., Senk, J., Hopkins, M., Schmidt, M., Stokes, A. B., et al. (2018). Performance comparison of the digital neuromorphic hardware SpiNNaker and the neural network simulation software NEST for a full-scale cortical microcircuit model. *Front. Neurosci.* 12, 291. doi:10.3389/fnins.2018.00291

van Meegen, A. and Lindner, B. (2018). Self-consistent correlations of randomly coupled rotators in the asynchronous state. *Phys. Rev. Lett.* 121, 258302. doi:10.1103/PhysRevLett.121.258302

van Vreeswijk, C. and Farkhooi, F. (2019). Fredholm theory for the mean first-passage time of integrate-and-fire oscillators with colored noise input. *Phys. Rev. E* 100, 060402

van Vreeswijk, C. and Sompolinsky, H. (1996). Chaos in neuronal networks with balanced excitatory and inhibitory activity. *Science* 274, 1724–1726. doi:10.1126/science.274.5293.1724

van Vreeswijk, C. and Sompolinsky, H. (1998). Chaotic balanced state in a model of cortical circuits. *Neural Comput.* 10, 1321–1371. doi:10.1162/089976698300017214

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., et al. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nat. Methods* 17, 261–272. doi:10.1038/s41592-019-0686-2

Wagatsuma, N., Potjans, T. C., Diesmann, M., and Fukai, T. (2011). Layer-dependent attentional processing by top-down signals in a visual cortical microcircuit model. *Front. Comput. Neurosci.* 5, 31. doi:10.3389/fncom.2011.00031

Wilson, H. R. and Cowan, J. D. (1972). Excitatory and inhibitory interactions in localized populations of model neurons. *Biophysical Journal* 12, 1 – 24. doi:http://dx.doi.org/10.1016/S0006-3495(72)86068-5

Wilson, H. R. and Cowan, J. D. (1973). A mathematical theory of the functional dynamics of cortical and thalamic nervous tissue. *Kybernetik* 13, 55–80. doi:10.1007/BF00288786