Waveform-relaxation methods for ordinary and stochastic differential equations with applications in distributed neural network simulations

Dissertation

Bergische Universität Wuppertal
Fakultät für Mathematik und Naturwissenschaften

eingereicht von
Jan Hahne, M. Sc.
zur Erlangung des Grades eines Doktors der Naturwissenschaften

Betreut durch Prof. Dr. Matthias Bolten

Wuppertal, 29.05.2018

Die Dissertation kann wie folgt zitiert werden:

urn:nbn:de:hbz:468-20180727-140556-2
[http://nbn-resolving.de/urn/resolver.pl?urn=urn%3Anbn%3Ade%3Ahbz%3A468-20180727-140556-2]

Acknowledgments

First of all, I would like to thank Matthias Bolten for raising my interest in the Human Brain Project in the first place and for giving me the opportunity to work in this project under his supervision. I am grateful to Andreas Frommer for enabling me to do my studies in his research group and for his generous help in the transitions between funding phases of the project.

I wish to sincerely thank Markus Diesmann and Moritz Helias for introducing me to the field of computational neuroscience and for suggesting several interesting topics for collaborations, which made this thesis possible. I have benefited greatly from the knowledge of David Dahmen and Jannis Schuecker on rate-based models and enjoyed the very pleasant working atmosphere during our several meetings.

I also wish to thank all my colleagues and former colleagues of the Applied Computer Science Group, many of whom became friends over the years. I particularly thank Sarah Huber for proofreading this thesis.

Finally, I wish to thank my family and friends for their continuous support.

Foreword

The work presented in this thesis is in parts based on the following publications:

- J. Hahne, M. Helias, S. Kunkel, J. Igarashi, M. Bolten, A. Frommer, and M. Diesmann, *A unified framework for spiking and gap-junction interactions in distributed neuronal network simulations.*, Front. Neuroinformatics, 9 (2015)

- J. Hahne, M. Helias, S. Kunkel, J. Igarashi, I. Kitayama, B. Wylie, M. Bolten, A. Frommer, and M. Diesmann, *Including gap junctions into distributed neuronal network simulations*, in Brain-Inspired Computing: Second International Workshop, BrainComp 2015, Cetraro, Italy, July 6-10, 2015, Revised Selected Papers, K. Amunts, L. Grandinetti, T. Lippert, and N. Petkov, eds., Springer International Publishing, Cham, 2016, pp. 43–57

  Parts of these publications are incorporated in Chapters 1 and 5.

- J. Hahne, D. Dahmen, J. Schuecker, A. Frommer, M. Bolten, M. Helias, and M. Diesmann, *Integration of continuous-time dynamics in a spiking neural network simulator*, Front. Neuroinformatics, 11 (2017)

  Parts of this publication are incorporated in Chapters 2 and 6. The neuroscientific applications in Subsec. 6.4.3 of this thesis have been developed by the coauthors David Dahmen and Jannis Schuecker.

Contents

Chapter 1

Introduction

From a mathematical point of view this thesis deals with the efficient solution of large systems of ordinary

$$y' = f(t, y)$$

and stochastic

$$dX = a(t, X)\,dt + b(t, X)\,dW$$

differential equations with waveform-relaxation techniques in a restricted distributed setting, where communication between predefined subsystems is only possible on an equidistant time grid. The size of the systems considered ranges from small systems with less than 100 equations up to systems with $3.15 \cdot 10^9$ equations.

From a more workflow-inspired point of view this thesis evolved from work in the Horizon 2020 FET Flagship Project "Human Brain Project" and deals with the inclusion of new features in spiking neural network simulators. This task ranges from the analysis of the already existing structures of these simulators to the determination, development and analysis of suitable mathematical methods for desired new features and their implementation into existing spiking neural network simulator code. Most of the results of this thesis have already been published in journal articles with a neuroscientific focus [72, 73, 74]. The corresponding implementations of the algorithms are available as open source software in the NEST simulator [14, 110, 145].

Parallel spiking neuronal network simulators distribute the neurons over the computation nodes. The parallelization makes use of the fact that the dynamics of

the neurons with chemical synapses is decoupled for the duration of the minimal network delay $d_{\min}$ and thus can be solved independently for this duration. The solver for the differential equations describing the single-neuron dynamics is specified on the single-neuron level and may be different for different cell types. The MPI communication between compute nodes happens collectively for all neurons on the node and only once for the duration of the minimal delay $d_{\min}$. These fundamental structural decisions are crucial for the performance of neuronal simulators and their scalability on supercomputers where communication is expensive because it is associated with considerable latency.

In this thesis we aim to include two fundamental new features in these simulators: electrical synapses, so-called *gap junctions*, and *rate models*, which describe neurons or entire populations of neurons in terms of continuous variables, e.g. firing rates. Creating a gap junction or a connection between two rate models causes a coupling of the corresponding single-neuron dynamics at all time, resulting in one inseparable larger system of differential equations which, in a realistic network simulation, contains the dynamics of all or almost all neurons.

Although there are solvers which are specialized to the distributed solution of very large systems of differential equations like PVODE [27] of the software package SUNDIALS [90], they cannot be employed in the context of distributed neuronal network simulations, due to the incompatible overall workflow: these solvers receive the entire system of differential equations as input and integrate the dynamics by some user specified numerical method. In the more common case of an implicit numerical method, the resulting system of nonlinear algebraic equations is either solved by fixed-point iteration or by Newton iteration. The latter requires the solution of a linear system of equations. The idea of parallelization is to distribute the system of ODEs over the available computation nodes such that each node is solving a contiguous subset of the system. This is achieved by correspondingly distributing all vector operations (e.g. dot products, the calculation of norms, and linear sums) over the computation nodes. Each node computes the local part of each vector operation followed, if necessary, by a global MPI reduce operation (see [27] for further details on CVODE). Thus, this software conceptually uses one instance of the employed ODE solver and distributes its vector computations across the computation nodes, which is an entirely different approach from that employed in spiking neural network simulators.

We therefore develop problem-specific waveform-relaxation methods that enable the solution of the entire systems of differential equations with an overall workflow that is compatible with the restrictions of spiking neural network simulators.

The structure of this thesis can be outlined as follows: first in Chapter 2 we present some basic material with the goal to make this thesis as self-contained as possible. This includes a short introduction to the numerical solution of ordinary

and stochastic differential equations and a more detailed introduction on spiking neural network simulators. Subsequently in Chapter 3, we introduce waveform-relaxation methods for ordinary differential equations and develop a method suitable for the integration of gap junctions into spiking neural network simulators. After a review of important previous results, we describe our method of choice and analyze its convergence. In this chapter, we follow a purely mathematical perception and provide descriptions in a universal manner. We do, however, of course consider the restrictions of our application. Chapter 4 follows the same logic but deals with waveform-relaxation methods for stochastic differential equations. Again, we first give an introduction to the corresponding field and then develop a suitable method for the integration of rate models in spiking neural network simulators. Afterwards, we derive important theorems which give insight into the convergence of the method and its convergence speed. Chapters 5 and 6 then present our applications. Both chapters start with a neuroscientific introduction motivating the new feature, followed by a detailed description of the developed framework that enables their incorporation in the NEST simulator. This includes practical details on the implementation of the waveform-relaxation methods introduced in Chapters 3 and 4. Both chapters comprise an extensive numerical results section, investigating the accuracy of the developed methods and the performance of the framework. The chapters also include a description of necessary changes to the user interface and a concluding discussion with neuroscientific focus. Finally, Chapter 7 discusses the results of this thesis from a more general point of view and gives an outlook on possibly interesting follow-up research.

Chapter 2

Review of basic material

This chapter gives an overview of basic material that is employed in the remainder of this thesis. First, we introduce ordinary differential equations (ODEs) and stochastic differential equations (SDEs) along with some selected numerical methods. The introductions are mainly based on the textbooks [75, 176] in the ODE- and [61, 103, 135] in the SDE-case and do not claim to be full introductions to the corresponding fields, but rather explain our notation and specific aspects that are used in the later developed methods.

Secondly, we turn to spiking neural network simulators, which are the main application for the methods developed in this thesis. Based on [19], we explain their general workflow and concepts. Then, we discuss a specific simulator, the NEST simulator, in more detail. Again, we skip some details and features of the simulators that are not important in the context of this thesis. For further reading on the NEST simulator, we refer to [147], which is also the main source for its introduction in this chapter.

2.1 Ordinary differential equations

An ordinary differential equation in its most general form reads

$$\frac{dy(t)}{dt} = y'(t) = f(t, y(t)). \tag{2.1}$$

If (2.1) satisfies an initial condition $y(t_0) = y_0$ it is called an initial value problem. From here on we consider systems of N ordinary differential equations

$$y'(t) = f(t, y(t)) \tag{2.2}$$

with initial condition $y(t_0) = y_0$. Here, $y(t) = (y_1(t), \ldots, y_i(t), \ldots, y_N(t))^T$ denotes an N-dimensional vector and $f : [t_0, t_0 + T] \times \mathbb{R}^N \to \mathbb{R}^N$ is an N-dimensional function.

A function y represents a solution of this initial value problem if and only if

$$y(t) = y_0 + \int_{t_0}^{t} f(s, y(s))\, \mathrm{d}s. \tag{2.3}$$

The Picard-Lindelöf theorem (see e.g. [75], their Chapter I.8) ensures the (local) existence and uniqueness of the solution, under the assumption that the function f is Lipschitz continuous with Lipschitz constant L, i.e. that f satisfies

$$\| f(t, x) - f(t, y) \| \;\leq\; L \, \| x - y \| \tag{2.4}$$

in some suitable norm for all $t \geq t_0$ and $x, y \in \mathbb{R}^N$.

2.1.1 Runge-Kutta methods

Most nonlinear ODEs cannot be solved analytically. We therefore rely on approximate numerical schemes to obtain the solution of a given initial value problem. We restrict our introduction to the class of Runge-Kutta methods.

Let Δt denote the (for now fixed) step size, $t_k = t_0 + k\Delta t$ the equidistant grid points of the discretization for $k = 0, \ldots, n$, and y_k the approximation for $y(t_k)$ obtained by some numerical method. Then, the class of Runge-Kutta methods can be defined in the following way:

Definition 2.1. Let s be an integer and let $A \in \mathbb{R}^{s \times s}$, $b \in \mathbb{R}^s$ and $c \in \mathbb{R}^s$ contain real coefficients. Then the method

$$y_1 = y_0 + \Delta t \sum_{l=1}^{s} b_l \, f(t_{1,l}^*, y_{1,l}^*) \tag{2.5}$$

with

$$\begin{aligned} t_{1,q}^* &= t_0 + c_q \Delta t \quad , \; q = 1, \ldots, s \\ y_{1,q}^* &= y_0 + \Delta t \sum_{l=1}^{s} a_{ql} \, f(t_{1,l}^*, y_{1,l}^*) \quad , \; q = 1, \ldots, s \end{aligned} \tag{2.6}$$

is called an *s-stage Runge-Kutta method* for the solution of (2.2).

The approximate solution for y_n can therefore be determined by repeatedly applying (2.5), i.e.

$$y_{k+1} = y_k + \Delta t \sum_{l=1}^{s} b_l \, f(t^*_{k+1,l}, y^*_{k+1,l}) \tag{2.7}$$

for $k = 0, \ldots, n-1$.

Remark 2.2. A Runge-Kutta method is uniquely determined by its coefficients, which can be written in a so-called Butcher-tableau

$$\begin{array}{c|c} c & A \\ \hline & b^T \end{array}$$

with $c \in \mathbb{R}^s$, $b \in \mathbb{R}^s$ and $A \in \mathbb{R}^{s \times s}$. For *explicit* Runge-Kutta methods A is a strictly lower triangular matrix and $c_1 = 0$.

Definition 2.3. A Runge-Kutta method (2.5) has *order* p if for sufficiently smooth problems (2.2), i.e. $f \in C^p$

$$\|y(t_0 + \Delta t) - y_1\| \leq K \Delta t^{p+1} \tag{2.8}$$

holds for some constant K.

Remark 2.4. For a p-order Runge-Kutta method

i) the Taylor series for the exact solution $y(t_0 + \Delta t)$ and y_1 coincide up to and including the term Δt^p.

ii) the local discretization error

$$\tau(\Delta t) = \frac{y(t_0 + \Delta t) - y_1}{\Delta t} \tag{2.9}$$

satisfies $\|\tau(\Delta t)\| = \mathcal{O}(\Delta t^p)$. Thus the method is consistent of order p.

iii) the global discretization error

$$e_n = y(t_n) - y_n \tag{2.10}$$

satisfies $\|e_n\| = \mathcal{O}(\Delta t^p_{\max})$, where $\Delta t_{\max} = \max(\Delta t_1, \ldots, \Delta t_n)$. Thus the method is convergent of order p.

In order to control the local error, we can employ an adaptive step size control. Let Δt_k denote the step size in step k and $t_k = t_{k-1} + \Delta t_k$ the corresponding grid point of the discretization for $k = 1, \ldots, n$. Our goal is to choose Δt_1 (and analogically all subsequent Δt_k) such that

$$| y_{1,i}^{(\Delta t_1)} - y_i(t_0 + \Delta t_1) | \ \leq \ \text{TOL} \tag{2.11}$$

holds for each component $i = 1, \ldots, N$ and some given error tolerance TOL.

If $y_{1,i}^{(\Delta t_1)}$ is computed with a Runge-Kutta method of order p, a second Runge-Kutta method of order $p+1$ can be used to obtain an estimator of order $p+1$ for the error (2.11) of the former method.

Embedded schemes can be applied to keep the additional computational effort for the estimator low. In this case, both methods only differ in the choice of the real coefficients b_1,..., b_s. Table 2.1 gives the Butcher tableau of the Runge-Kutta-Fehlberg 4(5) method, which will be employed at a later point in this thesis.

0						
$\frac{1}{4}$	$\frac{1}{4}$					
$\frac{3}{8}$	$\frac{3}{32}$	$\frac{9}{32}$				
$\frac{12}{13}$	$\frac{1932}{2197}$	$-\frac{7200}{2197}$	$\frac{7296}{2197}$			
1	$\frac{439}{216}$	-8	$\frac{3680}{513}$	$-\frac{845}{4104}$		
$\frac{1}{2}$	$-\frac{8}{27}$	2	$-\frac{3544}{2565}$	$\frac{1859}{4104}$	$-\frac{11}{40}$	
	$\frac{25}{216}$	0	$\frac{1408}{2565}$	$\frac{2197}{4104}$	$-\frac{1}{5}$	0
	$\frac{16}{135}$	0	$\frac{6656}{12825}$	$\frac{28561}{56430}$	$-\frac{9}{50}$	$\frac{2}{55}$

Table 2.1: **Runge-Kutta-Fehlberg 4(5)** Butcher tableau of the Runge-Kutta-Fehlberg 4(5) method.

2.2 Stochastic differential equations

Stochastic ordinary differential equations (SDEs) contain an additional random component, often referred to as *noise*. This noise is described by a *Wiener process*, also called Standard Brownian motion.

Definition 2.5. The Wiener process is a continuous stochastic process $(W(t), t \geq 0)$ that satisfies

i) $W(0) = 0$ almost surely,

ii) $W(t) - W(s) \sim \mathcal{N}(0, t-s)$ for all $0 \leq s \leq t$, i.e. $W(t) - W(s)$ is normally distributed with mean 0 and variance $t - s$,

iii) $W(t) - W(s)$ is independent of $W(r) - W(q)$ for all $0 \leq q < r < s < t$, i.e. $W(t) - W(s)$ is independent of the past values.

Remark 2.6. The Wiener process can be interpreted as the definite integral

$$\int_0^t \xi(t')dt' = W(t)\,,$$

of a Gaussian white noise $\xi(t)$ with $\mathbb{E}[\xi(t)] = 0$ and $\mathrm{Cov}[\xi(t), \xi(t')] = \delta(t - t')$. Here δ denotes the Dirac delta function. This is a paradox, as one can also show that $W(t)$ is not differentiable (see e.g. [61], their Chapter 3).

Again we consider initial value problems with initial condition $X(t_0) = X_0$. Note that here X_0 can also be a random number. $X(t)$ is the solution of a (for now one-dimensional) Itô-SDE if and only if it satisfies the corresponding stochastic integral equation

$$X(t) = X_0 + \int_{t_0}^t a(s, X(s))\, ds + \int_{t_0}^t b(s, X(s))\, dW(s)\,. \tag{2.12}$$

Here the second integral is an Itô integral

$$\int_{t_0}^t Y(s)\, dW(s) := \lim_{n\to\infty} \sum_{k=1}^n Y_{k-1} \cdot (W_k - W_{k-1})$$

with $Y_k = Y(t_0 + k \cdot \frac{t-t_0}{n})$ and $W_k = W(t_0 + k \cdot \frac{t-t_0}{n})$. We obtain a different kind of SDE, if the second integral is chosen as a Stratonovich integral, indicated by the symbol $\circ$,

$$\int_{t_0}^t Y(s) \circ dW(s) := \lim_{n\to\infty} \sum_{k=1}^n \frac{Y_{k-1} + Y_k}{2} (W_k - W_{k-1})$$

which approximates $Y(s)$ with the mid-point rule. Then the corresponding SDE is called a Stratonovich-SDE. In the case of additive noise ($b(t, X(t)) = b(t)$) the Itô and Stratonovich integrals coincide. Furthermore, if the noise is constant ($b(t, X(t)) = \sigma = \text{const.}$) the integrals can be solved analytically

$$\int_{t_0}^{t} \sigma \, dW(s) = \int_{t_0}^{t} \sigma \circ dW(s) = \lim_{n\to\infty} \sigma \cdot \sum_{k=1}^{n} (W_k - W_{k-1}) = \sigma \cdot (W(t) - W(t_0))$$

with $W(t) - W(t_0) \sim \mathcal{N}(0, t - t_0)$. We refer to [103] and [61] for a derivation and deeper discussion on the differences between the two types of stochastic integrals and focus only on Itô-SDEs in the remainder of this thesis.

As opposed to the case of ODEs, the differential notation corresponding to (2.12)

$$dX(t) = a(t, X(t))\, dt + b(t, X(t))\, dW(t) \tag{2.13}$$

only denotes an informal way of expressing the integral equation. There is, however, another widely used differential notation, called the Langevin form of the SDE, which is mostly employed in physics. It is motivated by the relation between the Wiener process and a Gaussian white noise $\xi(t)$ as described in Remark 2.6 and reads

$$\frac{dX(t)}{dt} = a(t, X(t)) + b(t, X(t))\, \xi(t)\,. \tag{2.14}$$

As the value of $\xi(t)$ at a certain point in time t cannot be defined as a real-valued random variable we prefer the representations (2.12) and (2.13) and will use them in the remainder of this thesis. It is, however, worth noticing that the stochastic integral equation corresponding to (2.14),

$$X(t) = X_0 + \int_{t_0}^{t} a(s, X(s))\, ds + \int_{t_0}^{t} b(s, X(s))\, \xi(s)\, ds$$

can be interpreted consistently with (2.12) as $dW(t) \equiv \xi(t)dt$.

From here on, we consider systems of N stochastic differential equations

$$dX(t) = a(t, X(t))\, dt + b(t, X(t))\, dW(t) \tag{2.15}$$

with initial condition $X(t_0) = X_0$. Here, $X(t) = (X_1(t), \ldots, X_i(t), \ldots, X_N(t))^T$ and $W(t) = (W_1(t), \ldots, W_i(t), , \ldots, W_N(t))^T$ denote N-dimensional vectors and $a : [t_0, t_0 + T] \times \mathbb{R}^N \to \mathbb{R}^N$ and $b : [t_0, t_0 + T] \times \mathbb{R}^N \to \mathbb{R}^{N\times N}$ are N and $N \times N$-dimensional functions respectively. $W(t)$ is an N-dimensional Wiener process, i.e., the components $W_i(t)$ are independent and identically distributed.

The existence and uniqueness of the solution $X(t)$ to (2.15) can be shown (see e.g. [135], their Chapter 5) under the assumption that

i) the functions a and b are Lipschitz continuous, i.e. that

$$\|a(t,x) - a(t,y)\| \; + \; \|b(t,x) - b(t,y)\| \; \leq \; L\,\|x - y\|$$

holds for some constant L and any $t \geq t_0$ and $x, y \in \mathbb{R}^N$,

ii) the functions a and b satisfy a linear growth condition, such that

$$\|a(t,x)\| \; + \; \|b(t,x)\| \; \leq \; K\,(1 + \|x\|)$$

for some constant K and any $t \geq t_0$ and $x \in \mathbb{R}^N$,

iii) the initial value X_0 is a random variable with finite variance defined on the same probability space as $W(t)$, i.e.

$$\mathbb{E}\left[\|X_0\|^2\right] < \infty\,.$$

2.2.1 Selected numerical methods

This section presents some basic numerical methods for SDEs that are employed in the remainder of this thesis.

Let again Δt denote the fixed step size, $t_k = t_0 + k\Delta t$ the grid points of the discretization for $k = 0, \ldots, n$, and X_k the approximation for $X(t_k)$ obtained by some numerical method, at which X_0 is the given initial value. Then strong convergence for numerical methods for SDEs can be defined in the following way:

Definition 2.7. A numerical method for SDEs is called *strongly convergent of order p* if the expected value of the global discretization error

$$\epsilon = \mathbb{E}\left[\,|X(t_n) - X_n|\,\right] \tag{2.16}$$

satisfies $\|\epsilon\| = \mathcal{O}(\Delta t_{\max}^p)$, with $\Delta t_{\max} = \max(\Delta t_1, \ldots, \Delta t_n)$.

2.2.1.1 Euler-Maruyama

The Euler-Maruyama method is a generalization of the Forward Euler method for ODEs. It approximates the integrands in (2.12) with their left-sided values accordingly.

Definition 2.8. The method

$$X_{k+1} = X_k + a(t_k, X_k) \cdot \Delta t + b(t_k, X_k) \cdot \Delta W_k \tag{2.17}$$

with $\Delta W_k = W(t_{k+1}) - W(t_k) \sim \mathcal{N}(0, \Delta t)$ for $k = 0, \ldots, n-1$ is called the *Euler-Maruyama method* for the solution of (2.13).

For general SDEs, the Euler-Maruyama method is strongly convergent with order 1/2. In the case of additive noise, it has strong convergence of order 1.

2.2.1.2 Semi-implicit Euler

The semi-implicit Euler method is a generalization of the backward Euler method for ODEs.

Definition 2.9. The method

$$X_{k+1} = X_k + a(t_{k+1}, X_{k+1}) \cdot \Delta t + b(t_k, X_k) \cdot \Delta W_k. \tag{2.18}$$

is called the *semi-implicit Euler method* for the solution of (2.13).

This method requires the solution of a system of nonlinear algebraic equations in every step. Basic techniques for the solution of the system are, e.g., Newton iteration or fixed-point iteration [101]. The method is called semi-implicit because the function b is still evaluated at (t_k, X_k) instead of (t_{k+1}, X_{k+1}). A fully implicit Euler scheme for SDEs is not practical, which can be observed from the simple one-dimensional test equation

$$dX(t) = aX(t)\,dt + bX(t)\,dW(t)$$

A fully implicit Euler scheme would imply

$$X_n = X_0 \cdot \prod_{k=1}^{n-1} \frac{1}{1 - a\Delta t - b\Delta W_k}$$

Some of the factors of this expression may become infinite depending on the specific realizations of ΔW_k. It can be shown that $\mathbb{E}(|X_n|)$ does not exist for this method (see e.g. [103], their Chapter 9.8). Thus the term *implicit Euler* usually refers to the semi-implicit method (2.18) and is used in the remainder of this thesis.

The implicit Euler method is strongly convergent of order 1/2 in the case of multiplicative noise and of order 1 for additive noise.

2.2.1.3 Exponential Euler

The exponential Euler method relies on the assumption that $a(t, X(t))$ consists of a linear part and a nonlinear remainder, i.e.,

$$a(t, X(t)) = A \cdot X(t) + f(t, X(t))$$

with $A \in \mathbb{R}^{N\times N}$. The idea is to solve the linear part exactly and to approximate the integral of the nonlinear remainder and the Itô integral with an Euler-like approach. Variation of constants for (2.15) yields

$$X(t) = e^{A(t-t_0)}X_0 + \int_{t_0}^{t} e^{A(t-s)} f(s, X(s))\, ds + \int_{t_0}^{t} e^{A(t-s)} b(s, X(s))\, dW(s)\,.$$

There are several versions of stochastic exponential Euler methods that differ in the approximation of the integrals. Unfortunately, a standardized nomenclature to distinguish the methods is so far missing. The simplest approach, sometimes named the stochastic Lawson-Euler scheme (e.g. in [106]), approximates the integrands with their left-sided values:

$$X_{k+1} = e^{A\Delta t}X_k + e^{A\Delta t} f(t_k, X_k)\cdot \Delta t + e^{A\Delta t} b(t_k, X_k)\cdot \Delta W_k\,.$$

More advanced schemes approximate the nonlinear part by keeping $f(s, X(s))$ constant for $[t_0, t)$ and solving the remaining integral analytically as

$$\begin{aligned}\int_{t_0}^{t} e^{A(t-s)} f(s, X(s))\, ds \;&\approx \int_{t_0}^{t} e^{A(t-s)} f(t_0, X(t_0))\, ds \\ &= A^{-1}(e^{A(t-t_0)} - I)\cdot f(t_0, X(t_0))\,.\end{aligned}$$

Here I denotes the $N \times N$ identity matrix. The same technique can be used for the Itô integral

$$\int_{t_0}^{t} e^{A(t-s)} b(s, X(s))\, dW(s) \approx \int_{t_0}^{t} e^{A(t-s)} b(t_0, X(t_0))\, dW(s)\,. \tag{2.19}$$

For a single SDE, Shoji [171] proposed a method where the remaining integral $\int_{t_0}^{t} e^{a(t-s)}\, dW(s)$ with $a \in \mathbb{R}$ is approximated by $\int_{t_0}^{t} \alpha\, dW(s)$, such that $\alpha \in \mathbb{R}$ is chosen to minimize the mean-square error. This results in a similar approximation as for the nonlinear part. Komori and Burrage [106] adapted this approach for systems of SDEs. The scheme reads

$$\begin{aligned}X_{k+1} = e^{A\Delta t}X_k &+ A^{-1}(e^{A\Delta t} - I)\cdot f(t_k, X_k) \\ &+ \frac{1}{\Delta t}\cdot A^{-1}(e^{A\Delta t} - I)\cdot b(t_k, X_k)\cdot \Delta W_k\,.\end{aligned}$$

Alternatively, calculating the variance of $X(t)$ within the approximation (2.19), amounts to

$$\begin{aligned}\mathrm{Var}\left(X(t)\right) &= b(t_0, X(t_0))^2 \cdot \mathrm{Var}\left(\int_{t_0}^{t} e^{A(t-s)}\, dW(s)\right) \\ &= b(t_0, X(t_0))^2 \cdot A^{-1}\left(\frac{e^{2A(t-t_0)} - I}{2}\right).\end{aligned}$$

The corresponding method proposed by [3] can be defined in the following way:

Definition 2.10. The method

$$\begin{aligned}X_{k+1} = e^{A\Delta t}X_k + A^{-1}(e^{A\Delta t} - I)\cdot f(t_k, X_k) \\ + \sqrt{A^{-1}\left(\frac{e^{2A\Delta t} - I}{2}\right)} \cdot b(t_k, X_k)\cdot \eta_k\end{aligned} \tag{2.20}$$

with $\eta_k \sim \mathcal{N}(0,1)$ is called the *exponential Euler method* for the solution of (2.13) with partially linear $a(t, X(t)) = A \cdot X(t) + f(t, X(t))$.

In the remainder of this thesis we exclusively employ (2.20) and just refer to it as the *stochastic exponential Euler scheme*. Numerical experiments in [3] indicate that the method is strongly convergent of order 1/2 for SDEs with multiplicative noise.

For more detailed reviews on the different stochastic exponential Euler methods available, we refer to [3] and [106].

2.2.2 Stochastic delay differential equations

If the function a (or b) of a SDE is not only dependent on $X(t)$, but also on the values of the function X at previous times, i.e.

$$dX(t) = a\left(t, X(t), X(t-d_1), \ldots, X(t-d_l)\right)\, dt + b(t, X(t))\, dW(t) \tag{2.21}$$

with positive constants $d_1, \ldots, d_l$, then the corresponding SDE is called *stochastic delay differential equation* (SDDE). Let $d_{\max} = \max(d_1, \ldots, d_l)$ denote the longest and $d_{\min} = \min(d_1, \ldots, d_l)$ the shortest delay. For SDDEs, the initial condition needs to be defined on the entire interval $[t_0 - d_{\max}, t_0]$, i.e. $X(t) = \upsilon(t),\ t \in [t_0 - d_{\max}, t_0]$, where $\upsilon : [t_0 - d_{\max}, t_0] \to \mathbb{R}^N$ is an N-dimensional function.

The solution of SDDEs is usually obtained in a stepwise manner: with the initial condition, we can calculate the solution on the interval $[t_0, t_0 + d_{\min}]$, which is then needed to obtain the solution on the interval $[t_0 + d_{\min}, t_0 + 2 \cdot d_{\min}]$ and so on.

The numerical schemes presented in Subsec. 2.2.1 can thus be analogously used for SDDEs in this stepwise manner, as long as all delays d_i are multiples of the fixed step size Δt.

2.3 Spiking neural network simulators

One strategy to describe networks of neurons in computational neuroscience is by modeling them with a spiking neural network model. This approach is motivated by the microscopic dynamics of individual neurons and is therefore called a *bottom-up approach*. The idea is to model the individual neuron dynamics and their interaction with each other. The individual neurons can be described as *hybrid systems*:

$$\frac{dy}{dt} = f(y(t)) \tag{2.22}$$

$$y(t) \leftarrow g_i(y(t)) \quad \text{upon spike from synapse } i \tag{2.23}$$

Here the state y of the neuron evolves continuously to some biophysical equations (2.22) and a spike received by the neuron triggers a change in some state variable y_j (2.23). The neuron emits a spike if its membrane potential V satisfies some threshold condition, e.g. $V > \theta$. Typically the membrane potential is taken as the first state variable, i.e. $y_1 = V$.

The focus of spiking neural network simulators ranges from detailed neuron morphology (*NEURON*: [29], *GENESIS*: [15]) to an abstraction of neurons without spatial extent (*NEST*: [14], *BRIAN*: [68]). In this thesis we will focus on the latter. In this case, the biophysical equations (2.22) of the individual neurons are ODEs (or SODEs), only dependent on the time t. Due to the missing spatial dimensions, the neurons in this setup are also called *point neurons*.

In networks of point neurons each chemical synapse has its own synaptic delay d_{ij}, which depends on the emitting neuron i and the receiving neuron j. These delays are the result of different biological processes which cannot be modeled more accurately for point neurons, e.g. the time the postsynaptic potential needs to travel from the synapse on a dendrite to the soma. Thus, a spike which is emitted at time t_1 in neuron i affects the connected neuron j at time $t_1 + d_{ij}$. In a network, the dynamics of all neurons is decoupled for the duration of the minimal network delay $d_{\min} = \min_{ij}(d_{ij})$. Hence, the dynamics of each neuron can be propagated independently for the duration $d_{\min}$ without requiring information from other neurons.

Efficient simulators make use of the delayed and point-event like nature of the spike interaction by distributing neurons across available processes and communicating spikes only after this period [129]. In the following subsection we explain setup and implementation of a specific simulator, the NEST simulator, in more detail.

2.3.1 NEST - NEural Simulation Tool

We start our introduction of NEST by quoting the following short descriptive summary of NEST from the simulator website `www.nest-simulator.org`:

"NEST is a simulator for spiking neural network models that focuses on the dynamics, size and structure of neural systems rather than on the exact morphology of individual neurons. [...] NEST is ideal for networks of spiking neurons of any size, for example:

- Models of information processing, e.g. in the visual or auditory cortex of mammals,
- Models of network activity dynamics, e.g. laminar cortical networks or balanced random networks,
- Models of learning and plasticity."

The simulator is implemented in object-oriented C++ code and provides a Python front-end, called PyNEST [51]. We will explain the functionality of NEST with the help of an example, which is given as a PyNEST script in Script 2.1.

Script 2.1: **Example network in the PyNEST syntax.** The example network consists of nine model neurons of type `iaf_neuron`, a Poisson spike generator and a `voltmeter` recording device. The script uses the syntax of the PyNEST interface of the NEST simulator as of version 2.12.0 [110].

```
1 import nest
2
3 # set number of virutal processes
4 nest.SetKernelStatus({'total_num_virtual_procs': 4})
5
6 # create spike generator, neurons and recording device
7 pg = nest.Create('poisson_generator')
8 n = nest.Create('iaf_neuron', 9)
9 vm = nest.Create('voltmeter')
10
11 # define sources and targets
12 s = ([pg[0], pg[0], n[0], n[0], n[1], n[2], n[2], n[3],
13       n[4], n[5], n[5], n[6], n[7], n[8], vm[0]])
14 t = ([n[0], n[2], n[1], n[5], n[3], n[3], n[4], n[6],
15       n[7], n[4], n[7], n[8], n[8], n[5], n[8]])
16
17 # create connections
18 nest.Connect(s, t, 'one_to_one', 'static_synapse')
19
20 # start simulation
21 nest.Simulate(50.)
```

A network model in NEST consists of three basic elements: *nodes*, *connections* and *events*, each of which is represented by an abstract base class. Neuron models described by hybrid systems inherit from class `Node` and implement the state vector, the internal dynamics (2.22), and the responses to different types of events (2.23). Other derived classes of base class `Node` include devices of any kind, such as recording devices or spike generators. NEST distributes all nodes over the available virtual processes. A virtual process is a thread that can arise either from multithreading via OpenMP [137] or MPI parallelization [124]. The total number of virtual processes N_{VP} is the number of MPI processes times the number of OpenMP threads per MPI process and can be specified by the user as demonstrated in line 4 of Script 2.1.

Each created node is assigned a unique integer `gid` that serves as a global identifier. The distribution of the nodes is done by a simple modulo operation: a node with global identifier `gid` is assigned to the virtual process `gid` mod N_{VP}. Thus, the order in which nodes are created influences the load-balancing of the simulator. Fig. 2.1 shows the nodes created for our example network along with their global identifier and the number of the virtual process they are assigned to.

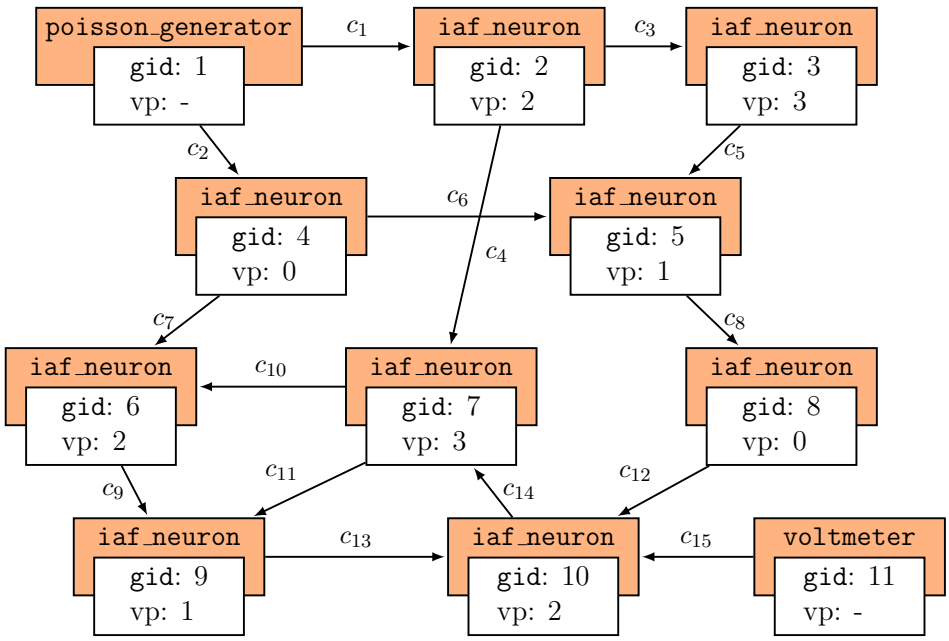

Figure 2.1: **Nodes and connections of the example network as a directed graph.** The orange boxes indicate the created nodes. The corresponding white boxes show their global identifier `gid` and the number of the virtual process they are assigned to in the simulation of Script 2.1 with four virtual processes. Nodes with "vp: -" are created for each virtual process. The edges c_i indicate the connections between the nodes with index according to the order in which they have been created.

All virtual processes living on the same MPI process share the same memory. Therefore, NEST creates a look-up table for each MPI process storing whether a specific node is a *local node*, and thus stored on the memory belonging to this MPI process, or a *proxy node*, and thus associated with another MPI process. Assuming our example is executed with two MPI processes $P_1 = \{\mathrm{vp}_0 \cup \mathrm{vp}_2\}$ and $P_2 = \{\mathrm{vp}_1 \cup \mathrm{vp}_3\}$, then the nodes with `gid`s $1, 2, 4, 6, 8, 10$ and 11 are local nodes on P_1, while the nodes with `gid`s $1, 3, 5, 7, 9$ and 11 are local nodes on P_2. The nodes with `gid`s 1 and 11 are local nodes to both MPI processes, as for devices, which often write data to memory, an own instance of the node is created for each virtual process.

Connections, which are the second basis element, inherit from the abstract base class `Connector`. They are used to model synaptic connections between neurons and to connect devices of any kind to neurons. All derived connection classes share the same four basic characteristics: a sending and a receiving node, specified by their `gid`s, and a weight and a delay, specified by positive real numbers.

Each virtual process only stores the incoming connections of its local nodes. This way, the memory requirements for the storage of the connections are kept on the lowest possible level. Each class derived from `Connector` stores all connections of this type with local target nodes in a suitable data structure. We again consider an execution of our example with two MPI processes. Then connections $c_1, c_2, c_7, c_8, c_{10}, c_{13}$ and c_{15} are known on P_1 but unknown on P_2, while the opposite holds for the remaining connections.

The events, the third basic element, are used to transmit spikes and other point events between connected nodes. Event classes are derived from the base class `Event` and only contain the `gid` of the sending node and the time at which the event occurred. Due to the massive amount of events, each virtual process first collects all events of its local nodes for the duration of the minimal network delay $d_{\min}$ and then communicates them all in one single MPI communication. For the communication across MPI processes, NEST uses `MPI_Allgather`; thus all events are communicated to all MPI processes and are available at every virtual process. After the communication, each virtual process goes through all events and checks if the events are associated with connections that have local target nodes. If this is the case, the time at which the event occurred is sent to the receiving neuron along with the weight and the delay of the connection. This way, the number of communications is kept to a very low level. As the amount of communicated data per spike is very low, the buffer sizes of the collective many-to-many communications are still negligible.

The actual implementation of the look-up table, the data structures to store the connections, and the communication of events is an active field of research and has been improved several times over the last years. The implementation of the

4th generation NEST kernel, which was the foundation for the work on NEST in this thesis, is described in [112] (for the very recently released 5th generation NEST kernel see [96]).

Another important concept of NEST is that all times are restricted to a time grid with fixed step size h. This means that every spike time and all information on the current state of a node is restricted to this grid, and that every delay of a synaptic connection has to be a multiple of h. Fig. 2.2 shows the progress of NEST over one interval of duration $d_{\min}$, the minimal network delay, from the single neuron perspective.

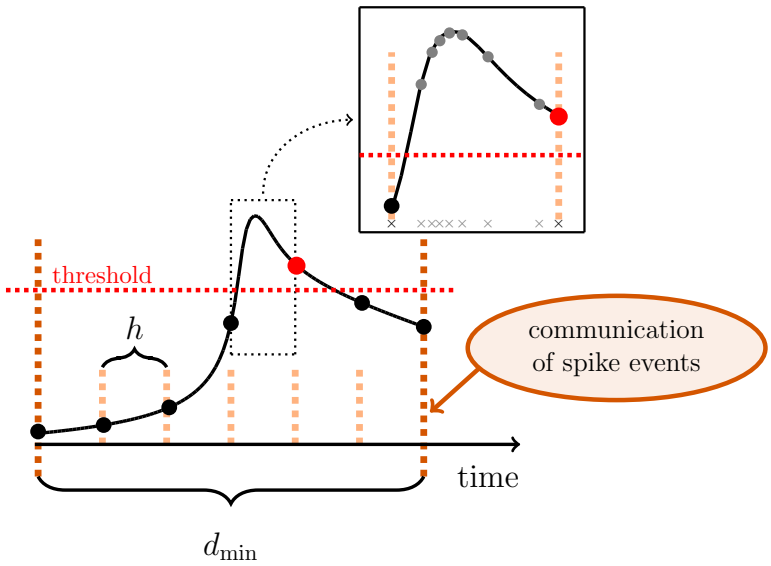

Figure 2.2: **Time course of the membrane potential of a single neuron during one $d_{\min}$-interval.** Black dots mark the values of the membrane potential at the grid points. The red dot indicates the point in time where the spike triggered by the threshold crossing is registered. The enlarged rectangle shows the gray marked internal grid points used by the model specific numerical method solving the neuron dynamics.

A numerical method solving the single neuron dynamics (2.22) is thus required to produce approximations of the state of the neuron at each grid point. This does, however, still allow the use of a method with adaptive step size within one interval of duration h. The state of the neuron at those additional evaluation points within the interval cannot, however, be recorded by any NEST device.

Chapter 3

Waveform-relaxation methods for ODEs

The term *waveform-relaxation methods* describes a set of iterative methods to solve systems of ordinary differential equations by dividing them into subsystems. The name and the concept were first introduced in the early 1980s by Lelarasmee, who employed a method of this kind for the simulation of large scale electric circuits [113, 114]. For any given initial value problem $y'(t) = f(t, y(t))$ with initial value $y(t_0) = y_0$, the basic idea is to divide the ODE-system into $v \leq N$ preferably weakly coupled subsystems

$$y_i'(t) = f_i(t, y_1(t), \ldots, y_v(t)) \quad i = 1, \ldots, v \tag{3.1}$$

and to solve each subsystem independently by treating the influence of the other subsystems as given input. For $v < N$ the resulting method is called a *block method*.

Starting with an initial guess $y_i^{(0)}(t)$ for the solution of each subsystem over the entire iteration interval $[t_0, t_0 + \mathcal{T}]$, the solution of the original ODE-system is determined by iteratively solving the independent subsystems, where for the i-th subsystem $y_1(t), ..., y_{i-1}(t), y_{i+1}(t), ..., y_v(t)$ is based on previously obtained solutions from the current or any of the previous iterations and hence acts as a given input to the i-th system. In order to fulfill the initial value condition, the initial guess $y_i^{(0)}(t)$ is usually chosen to be constant as $y_i^{(0)}(t) = y_0$ for all $t \in [t_0, t_0 + \mathcal{T}]$.

Two prominent examples of waveform-relaxation methods are

i) the (block) Jacobi waveform-relaxation method

$$y_i'^{(m)}(t) = f_i(t, y_1^{(m-1)}(t), ..., y_{i-1}^{(m-1)}(t), y_i^{(m)}(t), y_{i+1}^{(m-1)}(t), ..., y_v^{(m-1)}(t)) \tag{3.2}$$

with $i = 1, \ldots, v$, where the input for the m-th iteration is completely based on the solutions of the $(m-1)$-th iteration. For each iteration this strategy enables parallel processing of all subsystems and is hence well-suited for distributed simulations.

ii) the (block) Gauss-Seidel waveform-relaxation method

$$y_i'^{(m)}(t) = f_i(t, y_1^{(m)}(t), ..., y_{i-1}^{(m)}(t), y_i^{(m)}(t), y_{i+1}^{(m-1)}(t), ..., y_v^{(m-1)}(t)) \tag{3.3}$$

with $i = 1, \ldots, v$, where the input of subsystem i is taken from the current iteration for all systems $j < i$ and from the previous iteration for the remaining subsystems. Compared to the Jacobi version, this strategy is expected to speed up the convergence, but comes at the price of less potential for parallel processing.

Waveform-relaxation methods can be seen as an extension of the Picard-Lindelöf iteration

$$y'^{(m)}(t) = f(t, y^{(m-1)}(t)) \tag{3.4}$$

which plays an important part in the proof of the Picard-Lindelöf theorem (see Sec. 2.1) and was first mentioned almost one century before the first waveform-relaxation methods were developed [115]. Therefore some works, e.g. [131, 132], also refer to waveform-relaxation methods as the (generalized) *Picard-Lindelöf iteration*. Another quite commonly used term is *dynamic iteration*, e.g. in [11, 12]. This name explores the common ground between waveform-relaxation methods for linear ODEs and iterative methods for the solution of linear systems of algebraic equations (e.g. the Jacobi or Gauss-Seidel method), which in this view can be seen as the *static iteration* counterpart. Due to their setup, waveform-relaxation methods are particularly interesting for parallel simulations of large-scale problems. In the class of parallelizable methods, they can be classified as *parallel across the system* (see [26] for a review of parallelizable methods for ODEs).

Waveform-relaxation methods for ODEs and their convergence have been studied by various authors (for a quite extensive list see the introduction of [54]). In general one can distinguish between studies of *continuous* and *discrete* waveform-relaxation methods. The former investigate the application of the waveform-relaxation method directly to the continuous problem. The corresponding results are mostly of theoretical interest, as for most ODEs the solution cannot be obtained analytically. Therefore the solutions $y^{(m)}(t)$ are not available in continuous times but only at discrete grid points, depending on the used numerical method.

The latter studies, therefore, first apply some numerical method to the continuous formulation of the problem and investigate the convergence and behavior of the resulting waveform-relaxation method in comparison with the solution of the applied numerical scheme without the use of the waveform-relaxation technique.

In both cases, a formulation of the method with a so-called *splitting function* allows the simultaneous analysis of different waveform-relaxation methods. One considers the problem

$$y'^{(m)}(t) = F(t, y^{(m)}(t), y^{(m-1)}(t)) \tag{3.5}$$

where $F : [t_0, t_0 + T] \times \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}^N$ denotes the splitting function which fulfills

$$F(t, y, y) = f(t, y) \qquad \forall t \in [t_0, t_0 + T] \text{ and } \forall y \in \mathbb{R}^N . \tag{3.6}$$

The choice $F(t, y, z) = f(t, z)$ yields the Picard-Lindelöf iteration (3.4), while $F_i(t, y, z) = f_i(t, z_1, ..., z_{i-1}, y_i, z_{i+1}, ..., z_v)$ results in the block Jacobi ($v < N$) or Jacobi ($v = N$) waveform-relaxation method (3.2). The universal formulation also includes non-standard waveform-relaxation methods where a particular part of a function f containing $y_i(t)$ is evaluated with information from the current iteration, and the remaining part is evaluated with information from the previous iteration. For example, for the one-dimensional problem $f(t, y) = y^2$ the splitting function $F(t, y, z) = 0.7y^2 + 0.3z^2$ defines such a non-standard method.

In the following section we summarize the most important results from literature with respect to our application. Later, we develop our own method suitable for the inclusion of gap junctions in a spiking neural network simulator and analyze its convergence using already existing theorems from Bellen and Zerrano [10].

3.1 Literature review

For the continuous waveform-relaxation method we restrict our review to a theorem that has been stated by many authors (e.g. in [10, 12, 26, 95, 131]) and ensures the superlinear convergence of the method (3.5).

Theorem 3.1. *Let the splitting function F satisfy*

$$\begin{aligned} \|F(t, x, y) - F(t, \hat{x}, y)\|_2 &\leq K_1 \|x - \hat{x}\|_2 \qquad \forall x, \hat{x}, y \in \mathbb{R}^N,\ t \geq t_0 \\ \|F(t, x, y) - F(t, x, \hat{y})\|_2 &\leq K_2 \|y - \hat{y}\|_2 \qquad \forall x, y, \hat{y} \in \mathbb{R}^N,\ t \geq t_0 \end{aligned}$$

for some constants K_1 and K_2. Then the sequence $y^{(m)}(t)$, $m = 0, 1, \ldots$ produced by the waveform-relaxation method (3.5) *converges superlinearly to the solution $y(t)$ of the corresponding initial value problem, such that*

$$\sup_{t_0 \le t \le t_0+\mathcal{T}} \|y^{(m)}(t) - y(t)\|_2 \le e^{K_1 \mathcal{T}} \frac{(K_2 \mathcal{T})^m}{m!} \sup_{t_0 \le t \le t_0+\mathcal{T}} \|y^{(0)}(t) - y(t)\|_2 \,. \tag{3.7}$$

Proof. A general proof is included in [12] (see the proof of their Theorem 1.1). A proof for autonomous linear ODEs using an integral operator can be found in [131]. □

Remark 3.2. The former condition on F in Theorem 3.1 can be relaxed to a one-sided Lipschitz condition

$$(x - \hat{x})^T (F(t, x, y) - F(t, \hat{x}, y)) \;\le\; K_1 \|x - \hat{x}\|_2 \qquad \forall x, \hat{x}, y \in \mathbb{R}^N,\ t \ge t_0$$

which for some functions F allows for a negative constant K_1. This can significantly improve the error estimate.

For discrete waveform-relaxation methods the situation is more varied. First, the choice of the numerical method varies from study to study. While an early study by Nevanlinna [132] assumes a discretization with a linear multistep method, most authors use some kind of Runge-Kutta method for the discretization [9, 10, 95, 160]. Secondly in most studies the investigated systems of ODEs have specific properties that allow for new results. For example Bellen et al. [9] investigate systems that are dissipative in the maximum norm, Sand and Burrage [160] present an (in principle) universal method that is, however, only tested for linear systems with a tridiagonal matrix structure and in't Hout [95] analyzes the convergence for systems of stiff nonlinear ODEs.

Here, we restrict our review to studies that apply Runge-Kutta methods and do not pose any strong restrictions on the structure of the initial value problem. First, we state a basic theorem about the convergence of the discrete waveform relaxation using explicit Runge-Kutta methods formulated by, e.g., Bjørhus [12].

Theorem 3.3. *Let $y_k^{(m)}$ denote the approximate solution produced by the m-th iteration of the waveform-relaxation method that discretizes the continuous problem* (3.5) *with an explicit s-stage Runge-Kutta method* (2.5) *and let y_k denote the result of the same Runge-Kutta method applied to the corresponding initial value problem. Furthermore let F satisfy*

$$\begin{aligned} \|F(t, x, y) - F(t, \hat{x}, y)\|_2 &\le K_1 \|x - \hat{x}\|_2 \qquad \forall x, \hat{x}, y \in \mathbb{R}^N,\ t \ge t_0 \\ \|F(t, x, y) - F(t, x, \hat{y})\|_2 &\le K_2 \|y - \hat{y}\|_2 \qquad \forall x, y, \hat{y} \in \mathbb{R}^N,\ t \ge t_0 \end{aligned}$$

for some constants K_1 and K_2. Then the solution of the waveform-relaxation method with $\mathcal{T} = n \cdot \Delta t$ converges to the solution of the original Runge-Kutta scheme within $s \cdot n$ iterations such that for all $m \geq sn$

$$y_k^{(m)} = y_k \quad \forall k = 0, \ldots, n \,. \tag{3.8}$$

Proof. See [12] for the main arguments. □

Secondly, we turn to Bellen and Zerrano [10], who investigate the convergence of the discrete waveform-relaxation method using continuous Runge-Kutta methods. They considered the waveform-relaxation method

$$\begin{aligned} \tilde{y}^{(m)}(t_{k+1}) \;=\; & \tilde{y}^{(m)}(t_k) \\ & + \Delta t_{k+1} \sum_{l=1}^{s} b_l \, \tilde{F}\left(t^*_{k+1,l},\, y^{*(m)}_{k+1,l},\, \tilde{y}^{(m)}(t^*_{k+1,l}),\, \tilde{y}^{(m-1)}(t^*_{k+1,l})\right) \end{aligned} \tag{3.9}$$

with

$$y^{*(m)}_{k+1,q} \;=\; \tilde{y}^{(m)}(t_k) + \Delta t_{k+1} \sum_{l=1}^{s} a_{ql} \, \tilde{F}\left(t^*_{k+1,l},\, y^{*(m)}_{k+1,l},\, \tilde{y}^{(m)}(t^*_{k+1,l}),\, \tilde{y}^{(m-1)}(t^*_{k+1,l})\right)$$

for $q = 1, \ldots, s$. Here, $\tilde{y}^{(m)}(t)$ denotes the continuous approximation in the m-th iteration. Its values at the grid points $t_1, \ldots, t_n$ are determined by scheme (3.9), while the values $\tilde{y}^{(m)}(t_k + \theta \Delta t_{k+1})$ at intermediate points in time ($\theta \in (0,1)$) are interpolated by some kind of interpolation scheme. The extended splitting function $\tilde{F}$ is assumed to satisfy

$$\tilde{F}(t, y, y, z) = F(t, y, z) \qquad \forall t \in [t_0, t_0 + T] \text{ and } \forall y, z \in \mathbb{R}^N \,. \tag{3.10}$$

Bellen and Zerrano consider the particularly interesting case where different subsystems use different grid points $t^{(m)}_{1,i}, \ldots, t^{(m)}_{n_{(i,m)},i}$ which also change while iterating, thus they investigate a use of the method with an adaptive step size control and individual handling of different subsystems. They show that there exists a step size Δt^* such that (3.9) converges to some limit function $\tilde{y}(t)$, provided that

i) $\tilde{F}$ satisfies the Lipschitz conditions

$$\begin{aligned} \|\tilde{F}(t,x,y,z) - \tilde{F}(t,\hat{x},y,z)\|_2 &\leq K_1 \|x - \hat{x}\|_2 \qquad \forall x, \hat{x}, y, z \in \mathbb{R}^N,\ t \geq t_0 \\ \|\tilde{F}(t,x,y,z) - \tilde{F}(t,x,\hat{y},z)\|_2 &\leq K_2 \|y - \hat{y}\|_2 \qquad \forall x, y, \hat{y}, z \in \mathbb{R}^N,\ t \geq t_0 \\ \|\tilde{F}(t,x,y,z) - \tilde{F}(t,x,y,\hat{z})\|_2 &\leq K_3 \|z - \hat{z}\|_2 \qquad \forall x, y, z, \hat{z} \in \mathbb{R}^N,\ t \geq t_0 \end{aligned}$$

for some constants K_1, K_2, K_3,

ii) the employed interpolation scheme is included in a very general class of interpolation schemes (see [10], their equation (2.14)) that amongst others includes Lagrange and Hermite interpolation,

iii) the time meshes satisfy $\max_{k,i,m} \Delta t_{k,i}^{(m)} \leq \Delta t^*$ and some other basic assumptions, which are expected to be met by any reasonable step size control mechanism (see [10], their assumptions 2.4 and 2.5 and equation (2.18)).

In addition, they state the following theorem regarding the so-called *limit method* that generates the limit function $\tilde{y}(t)$:

Theorem 3.4. *If the Runge-Kutta method employed in* (3.9) *has order* p*, the (uniform) order of the interpolation procedure is* $q \geq p - 1$ *and the waveform-relaxation method* (3.9) *is convergent, then the limit method which generates the limit function* $\tilde{y}(t)$ *is of order* $\min(p, q)$*.*

Proof. See [10] for the outline of the proof. □

Remark 3.5. The limit method of scheme (3.9) is no longer a classical Runge-Kutta method. In [9] Bellen et al. investigate several limit methods that arise from discrete waveform-relaxation methods with Runge-Kutta methods and show that this type of implementation can in general improve the stability properties of the underlying Runge-Kutta methods.

3.2 An ODE-waveform-relaxation method suitable for spiking neural network simulators

With the knowledge of the above stated results in mind, we now describe the restrictions and requirements of our application and derive and analyze a suitable waveform-relaxation method.

3.2.1 Restrictions and requirements

In our application we consider systems of N ordinary differential equations that are arranged in many small homogeneous subsystems

$$y_i'(t) = f_i(t, y_1(t), \ldots, y_v(t)) \quad i = 1, \ldots, v \tag{3.11}$$

which are weakly coupled to each other. Each subsystem contains a couple of components, only one of which contributes to the solution of the other subsystems.

The number of subsystems v can be assumed to be of order $N/10$. The subsystems are distributed over all available compute nodes and expected to be solved simultaneously with an explicit method. Communication between the subsystems is only permitted on an equidistant time grid with step size h, but desired to occur only in larger intervals of length $d_{\min}$. We are thus able to apply a waveform-relaxation method successively on several intervals of length $\mathcal{T} = h$ or $\mathcal{T} = d_{\min}$. Depending on the current state y the subsystems (3.11) constitute stiff problems. The stiff behavior is, however, only observed in small time intervals.

3.2.2 The method

Because of the stiff behavior in small time intervals we decide against an explicit Runge-Kutta method with fixed step size h and instead follow the idea of Bellen and Zerrano, employing an explicit Runge-Kutta method with adaptive step size control to advance the state of the subsystems. In a distributed setting where communication is only permitted on a fixed time grid with step size h, the communication of interpolations between grid points $t_{1,i}^{(m)}, \ldots, t_{n_{(i,m)},i}^{(m)}$ is neither reasonable, due to the massive amount of communicated data, nor possible, due to the restrictions on the communication points. We therefore create our own method, which uses Hermite interpolation between potential communication points. In order to make our interpolation scheme possible we force the adaptive step size control to include the time points $t_0 + uh$, $u = 0, \ldots, \frac{d_{\min}}{h}$. Thus for each u, there exists some index k such that $t_{k,i}^{(m)} = t_0 + uh$. We denote this particular index by k_u. Due to the requirement of a parallel simulation, we choose a block Jacobi waveform-relaxation method. The final method reads

$$y_{k+1,i}^{(m)} = y_{k,i}^{(m)} + \Delta t_{k+1}^{(m)} \sum_{l=1}^{s} b_l \, f_i \Big(t_{k+1,i,l}^{*(m)}, \; \tilde{y}_1^{(m-1)}(t_{k+1,i,l}^{*(m)}), \ldots, \tilde{y}_{i-1}^{(m-1)}(t_{k+1,i,l}^{*(m)}), \; y_{k+1,i,l}^{*(m)}, \; \tilde{y}_{i+1}^{(m-1)}(t_{k+1,i,l}^{*(m)}), \ldots, \tilde{y}_v^{(m-1)}(t_{k+1,i,l}^{*(m)}) \Big) \tag{3.12}$$

with

$$y_{k+1,i,q}^{*(m)} = y_{k,i}^{(m)} + \Delta t_{k+1}^{(m)} \sum_{l=1}^{q-1} a_{ql} \, f_i \Big(t_{k+1,i,l}^{*(m)}, \; \tilde{y}_1^{(m-1)}(t_{k+1,i,l}^{*(m)}), \ldots, \tilde{y}_{i-1}^{(m-1)}(t_{k+1,i,l}^{*(m)}), \; y_{k+1,i,l}^{*(m)}, \; \tilde{y}_{i+1}^{(m-1)}(t_{k+1,i,l}^{*(m)}), \ldots, \tilde{y}_v^{(m-1)}(t_{k+1,i,l}^{*(m)}) \Big)$$

for $q = 1, \ldots, s$ and

$$\begin{aligned} \tilde{y}_i^{(m)}(t_0 + (u+\theta)h) \;=\; & y_{k_u,i}^{(m)} \cdot p_1(\theta) + y_{k_{u+1},i}^{(m)} \cdot p_2(\theta) + hf(t_0 + uh, \, y_{k_u,i}^{(m)}) \cdot p_3(\theta) \\ & + hf(t_0 + (u+1)h, \, y_{k_{u+1},i}^{(m)}) \cdot p_4(\theta) \end{aligned} \tag{3.13}$$

for $u = 0, \ldots, \frac{d_{\min}}{h} - 1$ and $\theta \in (0, 1)$. Here p_1, p_2, p_3 and p_4 are the Hermite basic polynomials, which read

$$p_1(\theta) = 1 - 3\theta^2 + 2\theta^3, \qquad p_2(\theta) = 3\theta^2 - 2\theta^3,$$
$$p_3(\theta) = \theta - 2\theta^2 + \theta^3, \qquad p_4(\theta) = -\theta^2 + \theta^3.$$

Remark 3.6. In comparison to (3.9) the continuous approximation $\tilde{y}_i^{(m)}$ of our method (3.12) discards the information of all grid points between $y_{k_u,i}^{(m)}$ and $y_{k_{u+1},i}^{(m)}$. This discarded information is, however, used beforehand for the computation of $y_{k_u,i}^{(m)}$ and $y_{k_{u+1},i}^{(m)}$. Therefore, the continuous approximation $\tilde{y}_i^{(m)}$ is more accurate than one obtained from a Hermite interpolation of a method with fixed step size h. This is due to the more reliable approximation of the exact solution at the grid points.

In Chapter 5, we employ method (3.12) with the Runge-Kutta-Fehlberg 4(5) method (see Table 2.1). We investigate the convergence of the method in the next section.

3.2.3 Convergence analysis

For convergence analysis, we make use of the fact that our method is closely related to (3.9). We argue why particular conclusions about the method from Bellen and Zerrano are also valid for our method (3.12), instead of repeating their technical and not very illustrative proof. First, we notice that our block Jacobi waveform-relaxation method, specified with the original function f, is included in the more general formulation with a splitting function F and that our time grids also satisfy the conditions formulated by Bellen and Zerrano, as they arise from a standard step size control mechanism. If we assume the same Lipschitz conditions for F (and thus for f), the only difference is located in the interpolation scheme. Although Hermite interpolation is also included in the class of interpolations considered by Bellen and Zerrano, their scheme ensures that $\tilde{y}_i^{(m)}(t_{k,i} + \Delta t_{k+1,i}) = y_{k+1,i}^{(m)}$ holds for each $k = 1, \ldots, n$. In our scheme, on the other hand, it only holds that $\tilde{y}_i^{(m)}(t_0 + uh) = y_{k_u,i}^{(m)}$ for each $u = 1, \ldots, \frac{d_{\min}}{h}$.

The proof of the convergence of method (3.9) to some limit function $\tilde{y}$ for sufficient small Δt^* (see [10], proof of their Theorem 2.2) is based on the contraction principle. The idea is to show that the sequence of mappings

$$\tilde{y}^{(m)} = \Phi^{(m)}(\tilde{y}^{(m-1)}), \quad m > 0, \quad \tilde{y}^{(0)} \text{ given},$$

is contractive on the space S of all continuous functions $\phi(t)$ of $[t_0, t_0+T]$ such that $\phi(t_0) = y_0$. In order to show that

$$\|\tilde{a}^{(m)} - \tilde{b}^{(m)}\| \overset{!}{<} \|\tilde{a}^{(m-1)} - \tilde{b}^{(m-1)}\|$$

with $\tilde{a}^{(m-1)}, \tilde{b}^{(m-1)} \in S$ holds for some suitable norm, one needs to find an estimate for the difference $\tilde{a}^{(m)} - \tilde{b}^{(m)}$ dependent on Δt^*. It is obvious that for our method, this difference can be bounded in the same manner, except that the estimate depends on h instead of Δt^*. The remaining proof can then be performed completely analogously. We thus conclude that our method also converges to some limit function $\tilde{y}$ if conditions i) and iii) from Sec. 3.1 are satisfied and h is sufficiently small.

The limit method which generates the limit function $\tilde{y}$ is defined by the iteration process and is described by (3.13) with the iteration index suppressed, i.e.

$$\begin{aligned}\tilde{y}_i(t_0 + (u+\theta)h) \;=\; & y_{k_u,i} \cdot p_1(\theta) \,+\, y_{k_{u+1},i} \cdot p_2(\theta) + \; hf(t_0 + uh,\, y_{k_u,i}) \cdot p_3(\theta) \\ & + hf(t_0 + (u+1)h,\, y_{k_{u+1},i}) \cdot p_4(\theta)\end{aligned} \tag{3.14}$$

for $u = 0, \ldots, \frac{d_{\min}}{h} - 1$ and $\theta \in (0,1)$. Here the $y_{k_u,i}$ are the values obtained with the underlying Runge-Kutta-method on the limit mesh

$$\begin{aligned}y_{k+1,i} \;=\; y_{k,i} + \Delta t_{k+1} \sum_{l=1}^{s} b_l\, f_i\,\big(t^*_{k+1,i,l},\;\; & \tilde{y}_1(t^*_{k+1,i,l}), \ldots, \tilde{y}_{i-1}(t^*_{k+1,i,l}), \\ & y^*_{k+1,i,l},\, \tilde{y}_{i+1}(t^*_{k+1,i,l}), \ldots, \tilde{y}_v(t^*_{k+1,i,l})\big)\end{aligned} \tag{3.15}$$

where

$$\begin{aligned}y^*_{k+1,i,q} \;=\; y_{k,i} + \Delta t_{k+1} \sum_{l=1}^{q-1} a_{ql}\, f_i\,\big(t^*_{k+1,i,l},\;\; & \tilde{y}_1(t^*_{k+1,i,l}), \ldots, \tilde{y}_{i-1}(t^*_{k+1,i,l}), \\ & y^*_{k+1,i,l},\, \tilde{y}_{i+1}(t^*_{k+1,i,l}), \ldots, \tilde{y}_v(t^*_{k+1,i,l})\big)\end{aligned}$$

for $q = 1, \ldots, s$.

Note that this is an implicit method across the intervals $[t_0 + uh, t_0 + (u+1)h]$, as all intermediate values between time points $t_{k_u,i}$ and $t_{k_{u+1},i}$ are dependent of $y_{k_{u+1},i}$ for all $u = 0, \ldots, \frac{d_{\min}}{h} - 1$.

It seems reasonable to expect that, if the underlying Runge-Kutta-method is of order p and the waveform-relaxation method is convergent, the error of $\tilde{y}$ with respect to the exact solution satisfies

$$\|y(t) - \tilde{y}(t)\| = \mathcal{O}(h^4) + \mathcal{O}(\Delta t^{*p}) \qquad \forall t \in [t_0, t_0+T]\;. \tag{3.16}$$

This assumption will be investigated in the numerical tests in Sec. 5.4.

Chapter 4

Waveform-relaxation methods for SDEs

Recently, researchers have started to investigate waveform-relaxation methods for stochastic differential equations. The main challenge of the extension to SDEs is to determine the influence of the additional stochastic part on the convergence of the well-established waveform-relaxation methods for ODEs. In addition, the stochastic part renders advanced approaches using interpolation (see, e.g. our method from Subsec. 3.2.2) infeasible: as the numerical solution of each time step involves a random number whose distribution depends on the step size Δt of the employed numerical method, waveform-relaxation methods for SDEs are only possible with a fixed step size. Maybe this is why, to our knowledge, there are so far very few publications on waveform relaxation for SDEs [53, 54, 55, 167].

Schurz and Schneider [167] derive sufficient conditions for the linear L^p convergence of the continuous waveform-relaxation methods for SDEs in a particular Banach space. Fan [54] presents a convergence proof for the continuous waveform-relaxation method for SDDEs and also proves the superlinear convergence of the method for a splitting function where the entire delayed part is taken from the previous iteration. The same author also investigates the convergence of the discrete waveform-relaxation method for SDDEs using a class of methods called semi-implicit Euler methods, which include the Euler-Maruyama and implicit Euler method [53].

In the following section we summarize the most important results with respect to our application. We then develop our own method suitable for usage in a spiking neural network simulator and analyze its convergence.

4.1 Literature review

In this section we state two theorems from the publications [53, 54] by Zhencheng Fan. The author considers stochastic delay differential equations with a fixed delay $d > 0$ and $t_0 = 0$ on a complete probability space $(\Omega, \mathcal{F}, \{\mathcal{F}_t\}_{t \geq t_0}, P)$ with a filtration $\{\mathcal{F}_t\}_{t \geq t_0}$:

$$\begin{aligned} dX(t) &= a\left(t, X(t), X(t-d)\right)\, dt + b(t, X(t), X(t-d))\, dW(t), \quad t \in (0, \mathcal{T}] \\ X(t) &= \upsilon(t), \quad t \in [-d, 0] \end{aligned} \tag{4.1}$$

where $\upsilon \in L^p_{\mathcal{F}_0}\left([-d, 0]; \mathbb{R}^N\right)$ holds for the initial function $\upsilon = \{\upsilon(t) : -d \leq t \leq 0\}$. Here, $L^p_{\mathcal{F}_0}\left([-d, 0]; \mathbb{R}^N\right)$ denotes the family of $\left(\mathcal{F}_0, \mathcal{B}\left(C\left([-d, 0], \mathbb{R}^N\right)\right)\right)$-measurable $C\left([-d, 0], \mathbb{R}^N\right)$-valued random variables $\xi = \{\xi(t) : -d \leq t \leq 0\}$ such that

$$\mathbb{E}\left[\sup_{-d \leq \theta \leq 0} \|\xi(\theta)\|_2^p\right] < \infty\,. \tag{4.2}$$

$C\left([-d, 0], \mathbb{R}^N\right)$ is the family of continuous functions from $[-d, 0]$ to $\mathbb{R}^N$ and $\mathcal{B}\left(C\left([-d, 0], \mathbb{R}^N\right)\right)$ denotes the Borel σ-algebra of C.

For the continuous waveform-relaxation method,

$$\begin{aligned} dX(t)^{(m)} &= A\left(t, X(t)^{(m)}, X(t)^{(m-1)}, X(t-d)^{(m-1)}\right)\, dt \\ &\quad + B\left(t, X(t)^{(m)}, X(t)^{(m-1)}, X(t-d)^{(m-1)}\right)\, dW(t), \quad t \in (0, \mathcal{T}] \\ X^{(m)}(t) &= \upsilon(t), \quad t \in [-d, 0] \end{aligned} \tag{4.3}$$

where the splitting functions A and B fulfill

$$A\left(t, X(t), X(t), X(t-d)\right) = a\left(t, X(t), X(t-d)\right)$$

$$B\left(t, X(t), X(t), X(t-d)\right) = b\left(t, X(t), X(t-d)\right)$$

for all $t \in [0, \mathcal{T}]$ and for all $X \in \mathbb{R}^N$. Fan shows the superlinear convergence of the method in the following theorem:

Theorem 4.1. *Let the splitting functions A and B satisfy*

$$\begin{aligned}(x-\hat{x})^T(A(t,x,y,z)-A(t,\hat{x},y,z)) &\leq K_1\|x-\hat{x}\|_2^2\\ \|A(t,x,y,z)-A(t,x,\hat{y},\hat{z})\|_2^2 &\leq K_2\|y-\hat{y}\|_2^2+K_3\|z-\hat{z}\|_2^2\\ \|B(t,x,y,z)-B(t,\hat{x},\hat{y},\hat{z})\|_2^2 &\leq K_4\|x-\hat{x}\|_2^2+K_5\|y-\hat{y}\|_2^2+K_6\|z-\hat{z}\|_2^2\end{aligned}$$

for some constants $K_1,\dots,K_6$ and any $t\geq 0$, $x,\hat{x},y,\hat{y},z,\hat{z}\in\mathbb{R}^N$. Additionally assume that $\upsilon\in L^2_{\mathcal{F}_0}$, i.e. $p=2$ in (4.1). Then the sequence $X^{(m)}(t)$, $m=0,1,\dots$ produced by the waveform-relaxation method (4.3) converges superlinearly to the solution $X(t)$ of (4.1), such that

$$\begin{aligned}&\sup_{0\leq s\leq t}\mathbb{E}\left[\|X^{(m)}(s)-X(s)\|_2^2\right]\\ &\qquad\leq e^{t(2K_1+K_4+1)}\frac{t^m}{m!}(K_2+K_3+K_5+K_6)^m\max_{0\leq s\leq t}\mathbb{E}\left[\|X^{(0)}(s)-X(s)\|_2^2\right]\end{aligned}\tag{4.4}$$

holds for every $t\in[0,\mathcal{T}]$

Proof. For the proof we refer to [54] (see the proof of their Theorem 3.1). □

Furthermore, Fan studies the convergence of the following discrete waveform-relaxation method that arises from the solution of (4.1) with the semi-implicit Euler method:

$$\begin{aligned}X_{k+1}^{(m)} = X_k^{(m)} &+ \theta\tilde{A}\left(t_{k+1},X_{k+1}^{(m)},X_{k+1}^{(m-1)},X_{k-\frac{d}{\Delta t}+1}^{(m)},X_{k-\frac{d}{\Delta t}+1}^{(m-1)}\right)\cdot\Delta t\\ &+(1-\theta)\tilde{A}\left(t_k,X_k^{(m)},X_k^{(m-1)},X_{k-\frac{d}{\Delta t}}^{(m)},X_{k-\frac{d}{\Delta t}}^{(m-1)}\right)\cdot\Delta t\\ &+\tilde{B}\left(t_k,X_k^{(m)},X_k^{(m-1)},X_{k-\frac{d}{\Delta t}}^{(m)},X_{k-\frac{d}{\Delta t}}^{(m-1)}\right)\cdot\Delta W_k\\ X_l^{(m)} = \upsilon(l\Delta t),&\quad l=-\frac{d}{\Delta t},\frac{d}{\Delta t}+1,\dots,0\,.\end{aligned}\tag{4.5}$$

The method requires $0\leq\theta\leq 1$ and obviously results in the implicit Euler method for $\theta=1$ and the Euler-Maruyama method for $\theta=0$. $\tilde{A}$ and $\tilde{B}$ denote the splitting functions which this time fulfill

$$\begin{aligned}\tilde{A}\left(t,X(t),X(t),X(t-d),X(t-d)\right) &= a\left(t,X(t),X(t-d)\right)\\ \tilde{B}\left(t,X(t),X(t),X(t-d),X(t-d)\right) &= b\left(t,X(t),X(t-d)\right)\end{aligned}$$

for all $t\in[0,\mathcal{T}]$ and for all $X\in\mathbb{R}^N$. Regarding the convergence of the method (4.5) Fan provides the following theorem:

Theorem 4.2. *Let $X_k^{(m)}$ denote the approximate solution produced by the m-th iteration of the waveform-relaxation method (4.5) and let X_k denote the solution of the non-iterative standard semi-implicit Euler scheme. Furthermore let the splitting functions $\tilde{A}$ and $\tilde{B}$ satisfy*

$$\begin{aligned}\|\tilde{A}(t,w,x,y,z) - \tilde{A}(t,\hat{w},\hat{x},\hat{y},\hat{z})\|_2^2 \\ \leq K\left(\|w-\hat{w}\|_2^2 + \|x-\hat{x}\|_2^2 + \|y-\hat{y}\|_2^2 + \|z-\hat{z}\|_2^2\right)\end{aligned}$$

and

$$\begin{aligned}\|\tilde{B}(t,w,x,y,z) - \tilde{B}(t,\hat{w},\hat{x},\hat{y},\hat{z})\|_2^2 \\ \leq K\left(\|w-\hat{w}\|_2^2 + \|x-\hat{x}\|_2^2 + \|y-\hat{y}\|_2^2 + \|z-\hat{z}\|_2^2\right)\end{aligned}$$

for some constant K and any $t \geq 0$, $w, \hat{w}, x, \hat{x}, y, \hat{y}, z, \hat{z} \in \mathbb{R}^N$. Additionally assume that $\upsilon \in L^1_{\mathcal{F}_0}$, i.e. $p = 1$ in (4.1). Then the solution of the waveform-relaxation method with $\Delta t = \mathcal{T}/n$ converges to the solution of the original scheme in the limit of $\Delta t \to 0$ and $m \to \infty$, i.e.

$$\lim_{\substack{m\to\infty \\ n\to\infty}} \max_{0\leq k\leq n} \mathbb{E}\left[\|X_k^{(m)} - X_k\|_2^2\right] = 0 \tag{4.6}$$

Proof. For the proof we refer to [53] (see the proof of their Theorem 2.1). □

4.2 A SDE-waveform-relaxation method suitable for spiking neural network simulators

With the knowledge of the above stated results in mind, we now once again describe the restrictions and requirements of our application, and derive and analyze a suitable waveform-relaxation method.

4.2.1 Restrictions and requirements

In our application we consider systems of N stochastic delay differential equations

$$dX(t) = \left[-A \cdot X(t) + f(t, X(t), X(t-d_1), \ldots, X(t-d_l))\right] dt + b(t)\, dW(t) \tag{4.7}$$

where $A = \mathrm{diag}(a_1, \ldots, a_N) \in \mathbb{R}^{N\times N}$ is a matrix with positive real coefficients a_i on the diagonal and zeros everywhere else. Thus, each equation contains a

self-dependent linear part, a nonlinear remainder with additional delayed coupling, with delays $d_1, \ldots, d_l \geq 0$, and additive noise, independent of $X(t)$. We are interested in a distributed simulation where each equation can be solved simultaneously. For our application the length of the iteration interval $\mathcal{T}$ of the waveform-relaxation method is, by design, less than or equal to the shortest of the involved delays d_i. Thus, the delayed terms $X(t - d_i)$ are the same in each iteration of the waveform-relaxation method.

We employ the waveform-relaxation method successively on several intervals of length $\mathcal{T}$. Thus, beginning at $t_0 + d_{\max}$ the initial function $\upsilon(t)$ for the current interval is given by the results from previous intervals. At the start of the simulation, $\upsilon(t)$ is zero on the interval $[t_0 - d_{\max}, t_0)$ for technical reasons and $\upsilon(t_0) = X_0$.

4.2.2 The method

For the sake of readability we assume from now on that all a_i are equal and denote them by a. It is easy to see that the following statements also hold for distinct a_i. Due to the requirement of a fully parallel simulation ($v = N$) we employ a Jacobi waveform relaxation

$$\begin{aligned} dX(t)^{(m)} &= \left[-aI \cdot X(t)^{(m)} + f(t, X(t)^{(m-1)}, X(t-d_1), \ldots, X(t-d_l))\right] dt \\ &\quad + b(t)\, dW(t) \end{aligned} \tag{4.8}$$

where in the m-th iteration the contributions of $X(t)$ to the nonlinear remainder are evaluated from the last iteration $m-1$, and the linear part is taken from the current iteration. The conditions for superlinear convergence of this continuous scheme are given by Theorem 4.1 in Sec. 4.1. Theorem 4.2 gives conditions under which the solution of (4.8) with the Euler-Maruyama or implicit Euler method converges to the non-iterative standard solution of these schemes in the limit of $\Delta t \to 0$ and $m \to \infty$. Due to the involved linear part we do, however, consider the stochastic exponential Euler method

$$\begin{aligned} X_{k+1}^{(m)} &= e^{-a\Delta t} X_k^{(m)} + \frac{1}{a}\left(1 - e^{-a\Delta t}\right) f(t_k, X_k^{(m-1)}, X_{k-\frac{d_1}{\Delta t}}, \ldots, X_{k-\frac{d_l}{\Delta t}}) \\ &\quad + \sqrt{\frac{1}{2a}(1 - e^{-2a\Delta t})} \cdot b(t_k) \cdot \eta_k \end{aligned} \tag{4.9}$$

for the solution which, in this setup, does not contain a matrix exponential or matrix square root due to the diagonal structure of A. The convergence of the discrete waveform-relaxation method using the stochastic exponential Euler method will therefore be investigated in the next section.

Remark 4.3. In this setup, the similarity of the exponential Euler and the Euler-Maruyama method

$$X_{k+1}^{\text{euler}} = X_k - a\Delta t X_k + f(t_k, X_k, \ldots)\Delta t + b(t_k) \cdot \Delta W_k$$

can be easily explored by expressing the exponential functions e^z with their defining power series $\sum_{j=0}^{\infty} \frac{z^j}{j!} = 1 + z + \frac{z^2}{2} + \ldots$. The exponential Euler scheme (4.9) can be written as

$$\begin{aligned}
X_{k+1}^{\text{exp}} &= X_k - a\Delta t X_k + \sum_{j=2}^{\infty} \frac{(-a\Delta t)^j}{j!} X_k \\
&\quad + \frac{1}{a}\left(1 - \left(1 + -a\Delta t + \sum_{j=2}^{\infty} \frac{(-a\Delta t)^j}{j!}\right)\right) f(t_k, X_k, \ldots) \\
&\quad + \sqrt{\Delta t - \frac{1}{2a}\sum_{j=2}^{\infty} \frac{(-2a\Delta t)^j}{j!}} \cdot b(t_k) \cdot \eta_k \\
&= X_k - a\Delta t X_k + f(t_k, X_k, \ldots)\Delta t + \sum_{j=2}^{\infty} \frac{(-a\Delta t)^j}{j!}\left(X_k - \frac{1}{a} f(t_k, X_k, \ldots)\right) \\
&\quad + \sqrt{\Delta t \left(1 + \sum_{j=2}^{\infty} \frac{(-2a\Delta t)^{j-1}}{j!}\right)} \cdot b(t_k) \cdot \eta_k
\end{aligned}$$

Together with $\sqrt{\Delta t} \cdot \eta_k = \Delta W_k$, we can calculate the difference between both schemes as

$$\begin{aligned}
\|X_{k+1}^{\text{exp}} - X_{k+1}^{\text{euler}}\|_2 &= \| \mathcal{O}(\Delta t^2)\left(X_k - \frac{1}{a} f(t_k, X_k)\right) \\
&\quad + \left(\sqrt{1 - a\Delta t + \mathcal{O}(\Delta t^2)} - 1\right) \cdot b(t_k) \cdot \Delta W_k \|_2
\end{aligned}$$

Thus, both schemes differ in the handling of both the deterministic and the stochastic part.

4.2.3 Convergence analysis

First, we want to derive a result similar to Theorem 4.2, but for our specific setup with the stochastic exponential Euler method and the choice of the Jacobi waveform-relaxation method. It turns out that the additive noise and the purely explicit method allow a statement without the expected value and with less restrictive conditions on m and Δt.

Theorem 4.4. *Let $X_k^{(m)}$ denote the approximate solution produced by the m-th iteration of the waveform-relaxation method (4.9), and let X_k denote the solution of the non-iterative standard exponential Euler scheme. Furthermore, let f satisfy*

$$\|f(t,x,z_1,\ldots,z_l) - f(t,y,z_1,\ldots,z_l)\|_2^2 < C \cdot \|x-y\|_2^2 \tag{4.10}$$

for some constant C and any $t \geq t_0$, $x, y, z_1, \ldots, z_l \in \mathbb{R}^N$. Then the solution of the waveform-relaxation method with $\mathcal{T} = n \cdot \Delta t$ converges to the solution of the original scheme within n iterations in the sense that for all $m \geq n$

$$\max_{0 \leq k \leq n} \|X_k^{(m)} - X_k\|_2^2 = 0\,. \tag{4.11}$$

Proof. Following (4.9), the difference between the waveform-relaxation method in iteration m and the original scheme is

$$\begin{aligned} X_{k+1}^{(m)} - X_{k+1} &= e^{-a\Delta t}\left(X_k^{(m)} - X_k\right) + \left(1 - e^{-a\Delta t}\right) \\ &\quad \Big(f(t_k, X_k^{(m-1)}, X_{k-\frac{d_1}{\Delta t}}, \ldots, X_{k-\frac{d_l}{\Delta t}}) \\ &\quad - f(t_k, X_k, X_{k-\frac{d_1}{\Delta t}}, \ldots, X_{k-\frac{d_l}{\Delta t}})\Big) \\ &=: e^{-a\Delta t}\alpha + \left(1 - e^{-a\Delta t}\right)\beta \end{aligned} \tag{4.12}$$

Accordingly,

$$\begin{aligned} \|X_{k+1}^{(m)} - X_{k+1}\|_2^2 &= e^{-2a\Delta t}\|\alpha\|_2^2 + \left(1 - e^{-a\Delta t}\right)^2 \|\beta\|_2^2 + 2e^{-a\Delta t} \cdot \left(1 - e^{-a\Delta t}\right)\alpha^T\beta \\ &\leq e^{-2a\Delta t}\|\alpha\|_2^2 + \left(1 - 2e^{-a\Delta t} + e^{-2a\Delta t}\right)\|\beta\|_2^2 \\ &\quad + \left(2e^{-a\Delta t} - 2e^{-2a\Delta t}\right) \cdot \left(\|\alpha\|_2^2 + \|\beta\|_2^2\right) \\ &= \left(2e^{-a\Delta t} - e^{-2a\Delta t}\right)\|\alpha\|_2^2 + \left(1 - e^{-2a\Delta t}\right)\|\beta\|_2^2\,. \end{aligned} \tag{4.13}$$

Using condition (4.10) we obtain

$$\begin{aligned}\|X_{k+1}^{(m)} - X_{k+1}\|_2^2 &\leq \underbrace{\left(2e^{-a\Delta t} - e^{-2a\Delta t}\right)}_{C_1}\|X_k^{(m)} - X_k\|_2^2 \\ &\quad + \underbrace{C \cdot \left(1 - e^{-2a\Delta t}\right)}_{C_2}\|X_k^{(m-1)} - X_k\|_2^2 .\end{aligned}$$

By defining $y_k^{(m)} = \|X_k^{(m)} - X_k\|_2^2$, this can be rewritten as

$$y_{k+1}^{(m)} \leq C_1 y_k^{(m)} + C_2 y_k^{(m-1)}$$

and (due to $y_0^{(m)} = 0$ for all $m \in \mathbb{N}$) with $y^{(m)} = (y_1^{(m)}, \ldots, y_n^{(m)})$ as

$$Py^{(m)} \leq Qy^{(m-1)} \tag{4.14}$$

respectively with $n \times n$-matrices

$$P = \begin{pmatrix} 1 & & & \\ -C_1 & \ddots & & \\ & \ddots & \ddots & \\ & & -C_1 & 1 \end{pmatrix} \quad \text{and} \quad Q = \begin{pmatrix} 0 & & & \\ C_2 & \ddots & & \\ & \ddots & \ddots & \\ & & C_2 & 0 \end{pmatrix} .$$

Rearranging (4.14) yields

$$\begin{aligned} y^{(m)} &\leq P^{-1}Qy^{(m-1)} \\ &\leq (P^{-1}Q)^m y^{(0)} . \end{aligned} \tag{4.15}$$

Now we apply the maximum norm to (4.15)

$$\begin{aligned}\|y^{(m)}\|_\infty &\leq \|(P^{-1}Q)^m y^{(0)}\|_\infty \\ &\leq \|(P^{-1}Q)^m\|_\infty \cdot \|y^{(0)}\|_\infty .\end{aligned}$$

We complete the proof by showing that $(P^{-1}Q)^n = 0$, i.e. that $P^{-1}Q$ is a nilpotent matrix. From

$$P^{-1} = \begin{pmatrix} 1 & & & & & \\ C_1 & 1 & & & & \\ C_1^2 & \ddots & \ddots & & & \\ C_1^3 & \ddots & \ddots & \ddots & & \\ \vdots & \ddots & \ddots & \ddots & \ddots & \\ C_1^{n-1} & \ldots & C_1^3 & C_1^2 & C_1 & 1 \end{pmatrix}$$

we obtain

$$P^{-1}Q = \begin{pmatrix} 0 & & & & & \\ C_2 & 0 & & & & \\ C_2C_1 & \ddots & \ddots & & & \\ C_2C_1^2 & \ddots & \ddots & \ddots & & \\ \vdots & \ddots & \ddots & \ddots & \ddots & \\ C_2C_1^{n-2} & \dots & C_2C_1^2 & C_2C_1 & C_2 & 0 \end{pmatrix}$$

which is a strictly lower triangular matrix and therefore nilpotent. □

Remark 4.5. Theorem 4.4 shows the convergence of scheme (4.9) for any step size $\Delta t > 0$. This convergence holds even for those step sizes where the exponential Euler method is unstable and yields inaccurate results. The number of iterations needed to obtain full convergence increases with decreasing step size Δt or increasing length $\mathcal{T}$ of the iteration interval.

Theorem 4.4 thus guarantees the convergence of the method and also provides an upper bound for the number of iterations that are required. It does not, however, provide any information on the evolution of the error over the iterations. Therefore, we follow a idea by Bjørhus, who derives error estimates for two discrete waveform-relaxation methods for ODEs that arise from the usage of the forward and the backwards Euler method [12]. As the proof is also based on the examination of the difference $\|X_{k+1}^{(m)} - X_{k+1}\|$, where the additive noise disappears from the equation (see (4.12)), we can easily adapt the idea for our method. In preparation, we state the following definition and specific discrete version of Gronwalls Lemma given in [12].

Definition 4.6. We denote by $z(n, \mathbb{R}^N)$ the space of all sequences $Z = \{Z_0, Z_1, \dots, Z_n\}$ where $Z_i \in \mathbb{R}^N$.

Lemma 4.7 (Gronwall)**.** *Let $Z, \Psi \in z(\infty, \mathbb{R})$ and $C > -1$. If*

$$Z_{k+1} - Z_k \leq CZ_k + \Psi_k \quad \textit{for } k = 0, 1, 2, \dots \tag{4.16}$$

then

$$Z_k \leq (1+C)^k \left(Z_0 + \sum_{j=1}^{k} (1+C)^{-j} \Psi_{j-1} \right) \quad \textit{for } k = 1, 2, 3, \dots \tag{4.17}$$

Now we can state the following result that provides an estimate for the error after m iterations with respect to the initial error.

Theorem 4.8. *Let again $X_k^{(m)}$ denote the approximate solution produced by the m-th iteration of the waveform-relaxation method (4.9) and X_k the solution of the non-iterative standard exponential Euler scheme. Furthermore, let $\|\cdot\|$ denote an arbitrary norm on $\mathbb{R}^N$ and f satisfy*

$$\|f(t,x,z_1,\ldots,z_l) - f(t,y,z_1,\ldots,z_l)\| < C \cdot \|x-y\| \tag{4.18}$$

in this arbitrary norm for some constant C and any $t \geq t_0$, $x,y,z_1,\ldots,z_l \in \mathbb{R}^N$. Then the error of the waveform-relaxation method with $\mathcal{T} = n\cdot\Delta t$ can be estimated for $m < n$ as

$$\begin{aligned} &\max_{0\leq k\leq n} \|X_k^{(m)} - X_k\| \\ &\qquad \leq \left(C\cdot(1-e^{-a\Delta t})\right)^m \binom{n}{m} e^{a\Delta t} \max_{0\leq k\leq n-m} \|X_k^{(0)} - X_k\|\,. \end{aligned} \tag{4.19}$$

Proof. In order to use Lemma 4.7 to derive an estimate for the sequence $\|X_k^{(m)} - X_k\| \in z(\infty,\mathbb{R})$, we first need to find an appropriate estimate for the difference $\|X_{k+1}^{(m)} - X_{k+1}\| - \|X_k^{(m)} - X_k\|$.

Since

$$\|X_{k+1}^{(m)} - X_{k+1}\| - \|X_k^{(m)} - X_k\| \leq \|X_{k+1}^{(m)} - X_{k+1} - \left(X_k^{(m)} - X_k\right)\|$$

we can estimate it by the right hand side term, which can be written as

$$\|\left(e^{-a\Delta t}-1\right)\left(X_k^{(m)} - X_k\right) + \left(1-e^{-a\Delta t}\right)\left(f(t_k, X_k^{(m-1)},\ldots) - f(t_k,X_k,\ldots)\right)\|\,.$$

Applying condition (4.18) thus yields the desired estimate

$$\begin{aligned} &\|X_{k+1}^{(m)} - X_{k+1}\| - \|X_k^{(m)} - X_k\| \\ &\qquad \leq \|\left(e^{-a\Delta t}-1\right)\left(X_k^{(m)} - X_k\right) + C\cdot\left(1-e^{-a\Delta t}\right)\left(X_k^{(m-1)} - X_k\right)\| \\ &\qquad \leq \left(e^{-a\Delta t}-1\right)\|X_k^{(m)} - X_k\| + C\cdot\left(1-e^{-a\Delta t}\right)\|X_k^{(m-1)} - X_k\|\,. \end{aligned} \tag{4.20}$$

From Lemma 4.7 we then get

$$\|X_k^{(m)} - X_k\| \leq C \cdot (1 - e^{-a\Delta t}) \sum_{j=1}^{k} (e^{-a\Delta t})^{k-j} \|X_{j-1}^{(m-1)} - X_{j-1}\| . \tag{4.21}$$

The recursive application of (4.21) yields

$$\begin{aligned} &\|X_k^{(m)} - X_k\| \\ &\qquad \leq \left(C \cdot (1 - e^{-a\Delta t})\right)^m \sum_{j_1=1}^{k} \sum_{j_2=1}^{j_1-1} \cdots \sum_{j_m=1}^{j_{m-1}-1} (e^{-a\Delta t})^{k-m-j_m} \|X_{j_m-1}^{(0)} - X_{j_m-1}\| . \end{aligned}$$

Next, we want to estimate the summands of the sum by its maximum. Since $e^{-a\Delta t}$ is real valued between 0 and 1, this factor is maximal if its exponent is minimal, i.e. if j_m takes its maximal value $k - m + 1$. Hence,

$$\begin{aligned} &\|X_k^{(m)} - X_k\| \\ &\qquad \leq \left(C \cdot (1 - e^{-a\Delta t})\right)^m \cdot e^{a\Delta t} \sum_{j_1=1}^{k} \sum_{j_2=1}^{j_1-1} \cdots \sum_{j_m=1}^{j_{m-1}-1} \|X_{j_m-1}^{(0)} - X_{j_m-1}\| . \end{aligned} \tag{4.22}$$

We now calculate the maximum over all $k \in 0, 1, \ldots n$ from the iteration interval.

$$\begin{aligned} &\max_{0 \leq k \leq n} \|X_k^{(m)} - X_k\| \\ &\qquad \leq \left(C \cdot (1 - e^{-a\Delta t})\right)^m \cdot e^{a\Delta t} \max_{0 \leq k \leq n-m} \|X_k^{(0)} - X_k\| \sum_{j_1=1}^{n} \sum_{j_2=1}^{j_1-1} \cdots \sum_{j_m=1}^{j_{m-1}-1} 1 \end{aligned} \tag{4.23}$$

Finally, we complete the proof by recognizing that the sum in (4.23) adds up ones for all unique, decreasing sequences of m integers between 1 and n, which is a descriptive definition of the binomial coefficient $\binom{n}{m}$. □

The insights of both theorems are employed in Chapter 6 in the context of integrating rate-based models in a spiking neural network simulator. While Theorem 4.4 is mostly used to guarantee accurate results and to justify using a waveform-relaxation method in this context, the error bound from Theorem 4.8 is employed in the numerical results in Subsec. 6.4.2 to explain and estimate the behavior for different kinds of rate-unit networks.

Chapter 5

Application in computational neuroscience I: Including gap junctions in a spiking neural network simulator

This chapter presents the first of our two major use cases for waveform-relaxation methods in computational neuroscience. Here we describe how to include gap junctions in a spiking neural network simulator.

Electrical synapses, or gap junctions, were classically regarded as a primitive form of neural signaling that played roles mostly in invertebrate neural circuits. Recently, advances in molecular biology revealed their widespread existence in the mammalian nervous system, such as visual cortex, auditory cortex, sensory motor cortex, thalamus, thalamic reticular nucleus, cerebellum, hippocampus, amygdala, and the striatum of the basal ganglia [34, 94], which suggests their diverse roles in learning and memory, movement control, and emotional responses [34, 45, 94]. The functional roles of gap junctions in network behavior are still not fully understood, but they are widely believed to be crucial for synchronization and generation of rhythmic activity. Recent results suggest that their contribution to synchronization is versatile, as the synchronization mediated by them depends on the intrinsic currents and morphology of the neurons as well as their interaction with inhibitory synapses [78]. A classification of this diversity of synchronization behaviors is addressed by the study of phase response curves [36, 78, 118], which describe a neuronal oscillator by its phase response to a perturbation. However, other prominent works also study more specific functional roles of gap junctions and combine the detailed simulation of small networks with experiments [181].

Even though brain-scale neural network simulations approach the size of the brain of small primates [88], and many biological features are already included, such

as the layer-specific connectivity and spike-timing dependent synaptic plasticity, simulations with correct cell densities are still lacking gap junctions. This is due to the absence of efficient algorithms to simulate gap junctions on large parallel computers.

Parallelized simulators for networks of spiking neurons like NEST suffer from conceptional difficulty in the handling of gap junctions, as they exploit the delayed and point-event like nature of the spike interaction between neurons: the dynamics of each neuron can be propagated independently for the duration $d_{\min}$ without requiring information from other neurons. This delayed communication scheme is crucial for the performance of the NEST simulator and its scalability to supercomputers, where communication is expensive, because it is associated with a considerable latency. Gap junctions, however, are typically represented by an instantaneous interaction between pairs of neurons of the form

$$I_{\text{gap},i}(t) = g_{ij}\ (V_j(t) - V_i(t)) \quad \text{and} \quad I_{\text{gap},j}(t) = -I_{\text{gap},i}(t)\ . \tag{5.1}$$

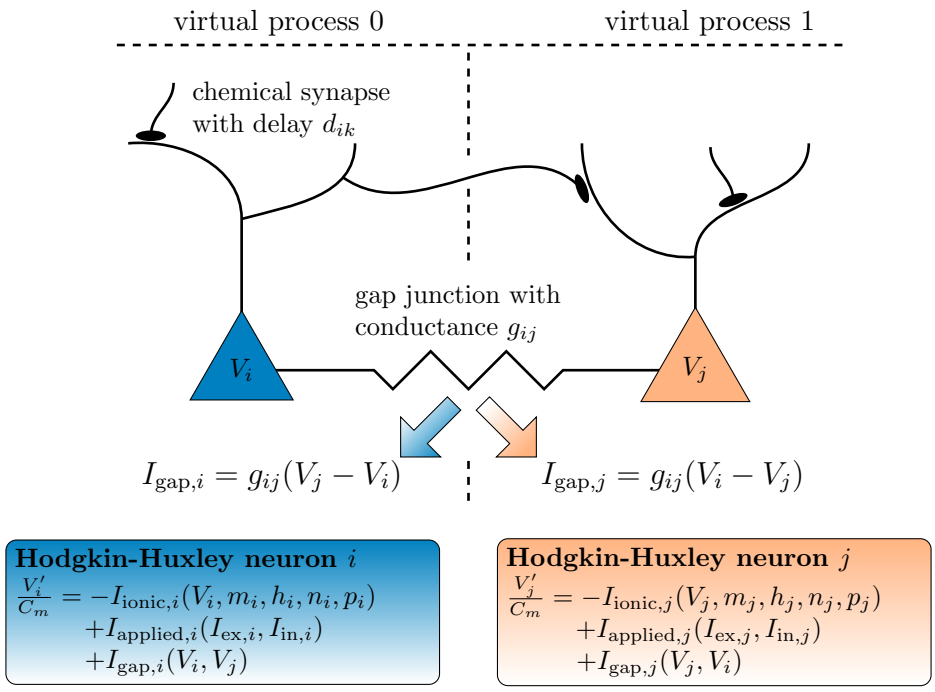

Figure 5.1: **Representation of two point neurons coupled by a gap junction.** The biophysical equation of the membrane potential refers to the later introduced new NEST model `hh_psc_alpha_gap`. The dashed lines indicate that neurons connected by a gap junction might be located on different virtual processes.

This current occurs in both cells at the site of the gap junction. In point-neuron models where equipotentiality is assumed the gap-junction current immediately

affects the membrane potential. Implementing a gap junction between neuron i and j in a time-driven simulation scheme therefore requires that neuron i knows the membrane potential of neuron j and vice versa at all times. The nature of the coupling between two neurons mediated by a gap junction depends on the difference of their membrane potentials; one neuron is excited, the other one inhibited.

Fig. 5.1 illustrates the effect of a gap junction on the systems of ODEs describing the single neuron dynamics. The originally decoupled hybrid systems of neurons i and j are combined as an interdependent system of ODEs. An additional gap-junction connection between another neuron and either i or j adds a further set of equations to the coupled system. In a biologically realistic simulation of a local cortical network, each neuron has dozens of gap-junction connections. Consequently, the dynamics of almost all neurons are likely interrelated by one large system of ODEs.

At present, time-driven simulators supporting gap junctions implement the instantaneous interaction with a simplification, effectively decoupling the neurons for the duration of the computation time step h: the membrane potentials of gap-junction coupled neurons are communicated at the beginning of each time step and for the purpose of integration are assumed to be constant for the duration of the time step (for NEURON simulation software see [93]). There is no communication for the duration of the computation time step and no repeated communication of improved membrane potential values for a given point on the computation-time grid. We refer to this approach as the *single-step method*.

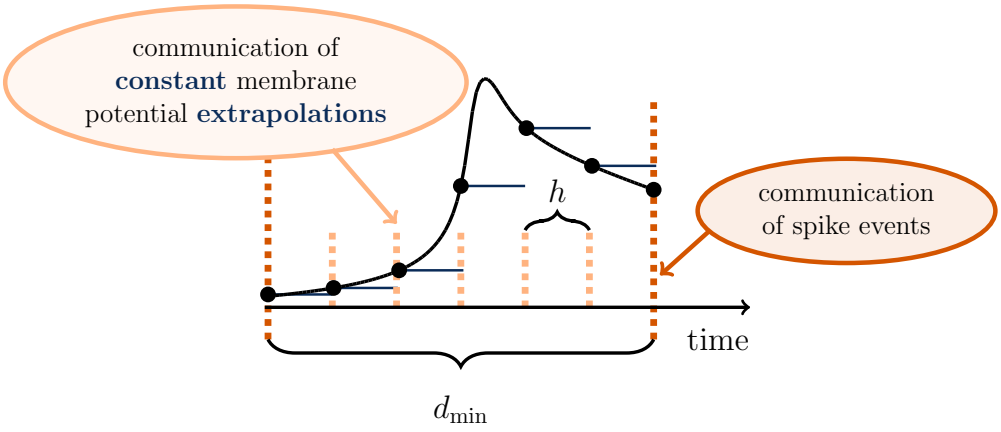

Figure 5.2: **Progress of the single-step method during one d_{min}-interval.** Black dots mark the values of the membrane potential at the grid points, which are communicated to the connected neurons after every interval of duration h.

Fig. 5.2 shows the progress of the single-step method over one interval of the duration of the minimal network delay d_{min} from the single neuron perspective.

Note that in this framework, a solver may still use adaptive step-size control to cover the interval h. The single-step method has two major disadvantages: First, communication is needed in every time step instead of in intervals of the minimal delay, which can slow down the simulation due to the latency of the employed MPI communication. Secondly the step size of the approach needs to be small. Otherwise the error causes a systematic shift of the membrane potential time course. This can be easily demonstrated by a two-neuron network: two identical model neurons that are coupled by a gap junction should behave exactly the same as an uncoupled pair since $I_{\text{gap}}(t) = 0$ holds at all times.

However, Fig. 5.3 shows that the single-step approach produces a significant shift within only 1 s of biological time, even when simulated with a step size that is already small compared to the time constants of the model neuron. This shift results from the calculation of I_{gap} at intermediate points in an interval of length h: As the membrane potential of the local neurons evolves over time while the membrane potential of the connected neuron stays constant an artefactual gap current $I_{\text{gap}} \neq 0$ introduces an error to the solution. To yield accurate results the time step would have to be exceedingly small, requiring even more communication and thereby further slowing down the simulation.

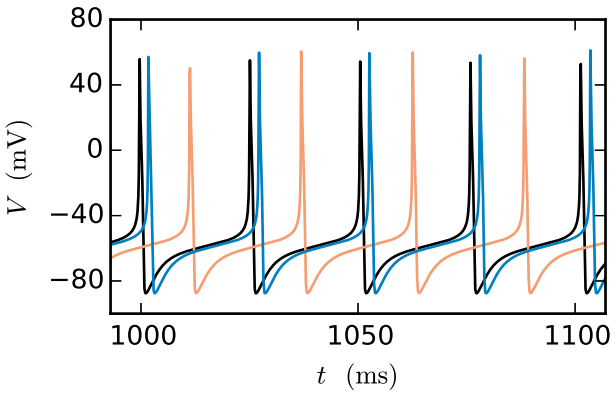

Figure 5.3: **Artefactual shift when using the single-step method.** The black curve shows the reference time course of the membrane potential of a Hodgkin-Huxley point-neuron model subject to a constant input current of 200 pA after simulating 1 s of biological time. The other curves indicate the time course of the membrane potential of the same neuron with the same input for the case that the neuron is coupled by a gap junction to a second model neuron with exactly the same properties, and the simulation is carried out with the single-step approach using a Runge-Kutta-Fehlberg solver with an adaptive step-size control to cover the interval of one computational time step h. The orange curve displays the result for a step size of $h = 0.1$ ms and the blue curve for 0.02 ms.

The iterative method based on waveform relaxation presented in Sec. 3.2 can be employed for the solution of the dynamics of spiking neural networks including gap junctions. It provides higher accuracy than the single-step implementation and additionally allows the continued usage of the delayed communication scheme. Sec. 5.1 describes how the approach can be smoothly integrated into the already existing structures of NEST as of version 2.8.0 [52], which uses the 4th generation kernel of NEST (see [112]). Then we provide details on the implemented neuron model (Sec. 5.2) and discuss required improvements of the user interface arising from the inclusion of gap junctions (Sec. 5.3). The following numerical results section comprises of four different subsections: the first subsection gives details on test cases, computers and measures of accuracy, Subsec. 5.4.2 and Subsec. 5.4.3 focus on the accuracy of the new iterative method in comparison to the single-step method and Subsec. 5.4.4 is concerned with the performance of the gap-junction framework in NEST. Finally, Sec. 5.5 discusses limitations and application areas. The technology described in this chapter is available as open source software under GNU General Public License (Version 2 or later). It was first released with NEST 2.10.0 [14]; several improvements have been added with NEST 2.12.0 [110]. Some of the figures and parts of the text in this chapter have already been used in the original research articles on the topic [73, 74].

5.1 Framework

The waveform-relaxation method described in Sec. 3.2 enables the efficient numerical solution of a system of ODEs on a parallel computing system, where each of the parallel processes is responsible for a particular subsystem. It therefore constitutes a promising way of implementing continuous electrical coupling between neurons through gap junctions in distributed simulations of neuronal networks. In this context, we identify the dynamics of the single neurons as the subsystems $y_i = (V_i, \ldots)$ and the membrane potentials from other neurons V_j as the influences from the other subsystems on system i. From here on we denote by N the total number of neurons (instead of the total number of components of a vector). Following Sec. 3.2 we consider the (block) Jacobi waveform-relaxation method

$$y_i'^{(m)}(t) = f_i(V_1^{(m-1)}(t), \ldots, V_{i-1}^{(m-1)}(t), y_i^{(m)}(t), V_{i+1}^{(m-1)}(t), \ldots, V_N^{(m-1)}(t)) \quad (5.2)$$

where the subsystems (with ≥ 1 component, depending on the neuron model) are solved with a Runge-Kutta method with adaptive step size control and the membrane potentials of the subsystems are interpolated on the intervals of length h between the grid points (see (3.12)). Subsec. 5.1.1 specifies the missing practical details regarding the management of the interpolation, the choice of the communication intervals and the iteration control of the employed waveform-relaxation method. To implement the method in a spiking neural network simulator, however, the simulator also needs to provide adequate infrastructure: gap-junction coupled neurons need to repeatedly update their state variables until a certain accuracy criterion is fulfilled. This implies that the scheduler of the simulator needs to support the repetition of neuronal updates. Subsec. 5.1.4 addresses this issue in more detail. Beforehand Subsec. 5.1.2 and Subsec. 5.1.3 describe the necessary changes to the fundamental data structures and discuss the potential impact of these changes on run time and memory consumption. During the design of the novel framework, we also kept in mind its potential for later extensions (one of which is presented in Chapter 6).

5.1.1 Algorithmic and numerical implementation

Applying the waveform-relaxation method specified in Sec. 3.2 to the current problem requires i) conveying the interpolation of the membrane potential of one neuron to another, ii) the definition of a communication protocol, in particular, the time points when information is exchanged between neurons and iii) the definition of an error estimate of the solution, to be used as a stopping criterion. These three points are described in the current section.

5.1.1.1 Interpolation of the membrane potential

The application of method (3.12) in context (5.2) requires a cubic Hermite interpolation of the membrane potentials V_i. For our numerical tests we are however also interested in investigating the method with different interpolations for the membrane potential. We thus also consider a simple constant interpolation and a linear interpolation. If we denote the interpolation of order n_{order} in the time interval $[kh, (k+1)h]$ as

$$V_i((k+\theta)h) = \sum_{s=0}^{n_{\text{order}}} a_{i,s}\theta^s \tag{5.3}$$

with $\theta \in [0,1]$, the coefficients of the interpolation polynomial can be determined as stated in Table 5.1.

n_{order}	$a_{i,0}$	$a_{i,1}$	$a_{i,2}$	$a_{i,3}$
0	V_0			
1	V_0	$V_1 - V_0$		
3	V_0	hV_0'	$-3V_0 + 3V_1 - h(2V_0' + V_1')$	$2V_0 - 2V_1 + h(V_0' + V_1')$

Table 5.1: **Coefficients of the interpolation polynomial depending on the interpolation order.** The entries use the following abbreviation: $V_0 \equiv V_i(kh)$, $V_1 \equiv V_i((k+1)h)$, $V_0' \equiv f(V_i(kh))$ and $V_1' \equiv f(V_i((k+1)h))$.

Table 5.2 shows the communication and storage demands generated by the use of the different interpolation orders. The communication effort is hardly surprising, since each neuron has to communicate the coefficients of the approximating polynomial to the gap-junction coupled neurons. Even though each neuron usually has multiple gap-junction connections, the incoming interpolation coefficients can be summed. If we denote the interpolation of the membrane potential V_j in the time interval $[kh, (k+1)h]$, as in (5.3), the total gap current $I_{\text{gap},i}$ reads:

	number of values to...	
interpolation	communicate per time step	store per time step
constant	1	2
linear	2	3
cubic	4	5

Table 5.2: **Storage and communication demand for different interpolation orders.** The storage and communication demands only differ by the sum of the gap weights g_{ij}, which needs to be stored in order to be able to sum up the incoming gap-junction connections.

$$\begin{aligned} I_{\text{gap},i} &= \sum_j g_{ij}(V_j - V_i) \\ &= -V_i \cdot \sum_j g_{ij} + \sum_j g_{ij} \cdot V_j \\ &= -V_i \cdot \sum_j g_{ij} + \sum_j g_{ij} \cdot \sum_{s=0}^{n_{\text{order}}} a_{j,s}\theta^s. \end{aligned} \tag{5.4}$$

It is obvious that within this time step, the effect of all incoming gap junctions can be reduced to the $n_{\text{order}} + 2$ parameters

$$\bar{g}_i := \sum_j g_{ij} \quad \text{and} \quad g^*_{i,s} := \sum_j g_{ij} \cdot a_{j,s}\ ,\ s = 0, \ldots, n_{\text{order}}.$$

The sums appearing in $\bar{g}_i$ and $g^*_{i,0}, \ldots, g^*_{i,n_{\text{order}}}$ over the connected neurons j can hence be performed once for each iteration step, and the total gap current (5.4) in each time step takes the form

$$I_{\text{gap},i} = -\bar{g}_i V_i + \sum_{s=0}^{n_{\text{order}}} g^*_{i,s}\theta^s. \tag{5.5}$$

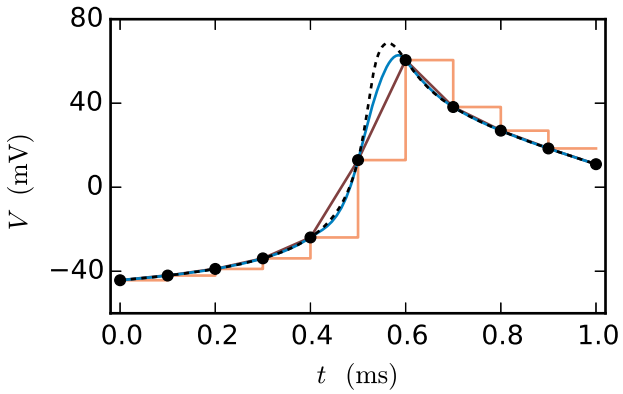

Figure 5.4: **Approximations of the membrane potential.** The dashed black curve shows the membrane potential representing an action potential (spike) and the black dots indicate the grid points used for the interpolation (step size 0.1 ms). The displayed interpolations are: piecewise constant (orange), linear (dark-red) and cubic (blue).

Fig. 5.4 shows the fit of the different interpolations for an exemplary course of the membrane potential. The piecewise constant interpolation is obviously a bad fit for the membrane potential. The choice between linear and cubic is investigated in Subsec. 5.4.2. Due to the fact that the linear interpolation requires less computation and has a lower communication and storage effort it could possibly achieve a better accuracy/simulation time trade-off.

5.1.1.2 Communication strategy

There are two different ways to use the waveform-relaxation method (3.12) in a time-driven simulator, as illustrated in Fig. 5.5.

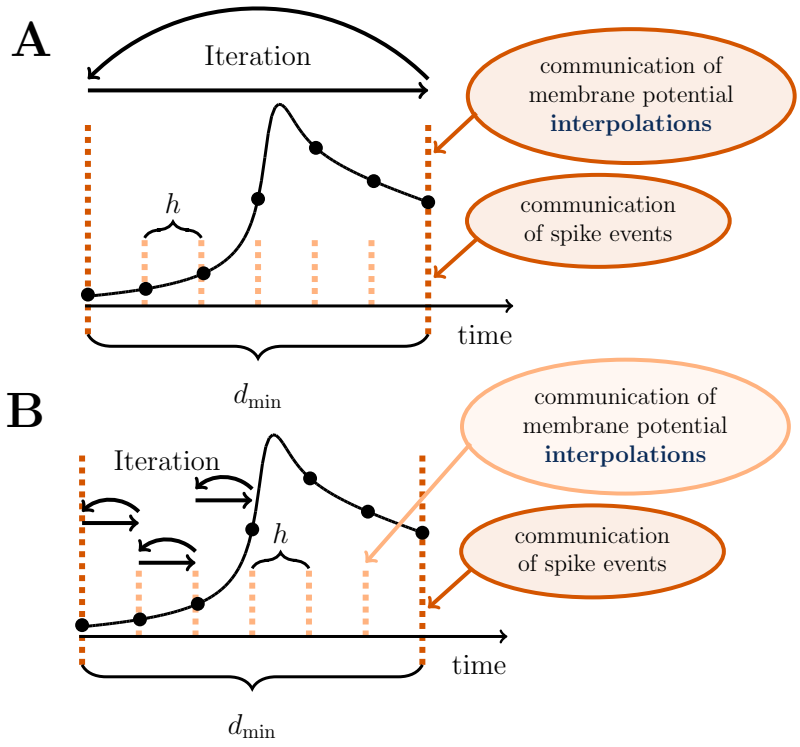

Figure 5.5: **Two communication strategies using the waveform-relaxation technique in a time-driven simulator. (A)** The membrane potentials are communicated in intervals equal to the minimal synaptic delay. **(B)** Potentials are communicated in every computation time step h.

The choice between the strategies is simply a question of the simulation time, since – given the convergence of the method – both strategies deliver nearly identical results for a given set of parameters. In this context it is worth noticing that the general convergence of the method does not depend on the length $\mathcal{T}$ of the iteration interval (see the analysis in Subsec. 3.2.3). The convergence speed, however, is expected to be much faster for shorter $\mathcal{T}$, as indicated by (3.7) in Theorem 3.1. The decoupling of the neurons for the duration of the minimal delay is exploited by iterating over the entire interval with minimal delay length, i.e. $\mathcal{T} = d_{\min}$. This approach allows us to keep the benefit of only one communication per minimal delay and is therefore expected to achieve the shorter simulation time - especially for simulations on large supercomputers, where the communication latency is important. The other obvious choice for the duration of an iteration

is $\mathcal{T} = h$. Due to the shorter iteration intervals, the latter choice is expected to need fewer iterations per time step and could for that reason be beneficial. We denote this strategy as *h-step communication* or *communication in every step*. Both communication strategies are investigated in Subsec. 5.4.4.

5.1.1.3 Iteration control

The convergence speed of the waveform-relaxation method (3.12) is dependent on multiple parameters. First, and most intuitively, the gap weights g_{ij} have an important influence, since they determine the coupling strength between the neurons that constitute the subsystems. It is obvious that stronger interaction causes slower convergence speed, since the extra iterations are only needed due to the external influences. Another important influence is the duration $\mathcal{T}$ of the iteration interval, as suggested by the error estimate (3.7) for the continuous waveform relaxation. This means that depending on the chosen communication strategy, either the choice of h (for $\mathcal{T} = h$) or the value of the minimal delay (for $\mathcal{T} = d_{\min}$) has an influence on the number of iterations needed. As a consequence of these multiple influences, the number of necessary iterations to achieve a certain accuracy may differ depending on the network being simulated and may be hard to determine for the user of the simulator. We therefore employ an adaptive iteration control which guarantees a certain accuracy, while avoiding unnecessary iterations. We introduce a new parameter `prelim_tol` and stop iterating if

$$|V_i^{(m)}(t_k) - V_i^{(m-1)}(t_k)| \leq \texttt{prelim\_tol}$$

holds for every grid point $k = 1, \ldots, n$ of every neuron $i = 1, \ldots, N$ within the iteration interval, or if the maximal number of iterations (`max_num_prelim_iterations`) is reached. The choice of those parameters is up to the user. The default settings are 10^{-4} for the `prelim_tol` and 15 for `max_num_prelim_iterations`. This kind of stopping criterion guarantees that if the first convergence criterion is met, further iteration would only improve the solution within the given error bound. The second criterion can be used to limit the computation time at time points that show slow convergence. If the iteration process is terminated by the second criterion, the user is notified by a warning. That way, the user can identify if the setting of the maximal number of iterations is adequate for the simulation. The `max_num_prelim_iterations` parameter should be increased if the maximal number of iterations is reached in more than a few iteration intervals.

The convergence control does not, however, protect from inaccuracies caused by the interpolated membrane potential $V_j^{(m-1)}(t)$ of the other neurons. Therefore,

the choice of h is also relevant for the communication strategy with $\mathcal{T} = d_{\min}$, even if it does not influence the number of iterations for this particular strategy.

5.1.2 Connection infrastructure

In the context of adaptations of the simulation kernel to current supercomputers, the connection infrastructure of NEST has undergone major changes to reduce the memory usage. The state-of-the-art prior to the inclusion of gap junctions is described in [112]. In NEST, connection objects are stored on the machine that hosts the target neuron of the particular connection. The corresponding data structure is required on each thread to provide efficient access to local connection objects of a given source neuron during event delivery (filled pink and turquoise squares in Fig. 5.6). Previously, these data structures were tailored to the delivery of spike events to local targets. The redesign presented here still supports the delivery of these events as described in [111] and [112] without compromising on performance. The delivery of data to mediate gap-junction coupling is different from the exchange of spiking activity in two respects: first, gap junctions require us to convey interpolation parameters of the membrane potentials from the sending to the receiving neuron. Secondly, the mechanism of data exchange should be generalizable, i.e. it should not be restricted to the implementation of gap junctions, but also applicable to other forms of interaction that require the exchange of data between neurons. The latter point implies the need to distinguish different connection types and events, called *secondary connections* and *secondary* or *payload events*, respectively. In contrast we will call spiking events *primary events*. We decided on a one-to-one correspondence between a secondary synapse type and the type of secondary event that can be sent via such a connection: A secondary event of a given type will be delivered only to the targets that are connected by the matching synapse type. The concrete implementation of gap junctions requires the definition of the new connection object `GapJunction` that is derived from the `Connection` base class, as well as a class `GapJEvent` that is derived from `SecondaryEvent`.

The extended connection infrastructure shown in Fig. 5.6A enables the storage of secondary connections. The parts of the structure that are new compared to Fig. 3 in [112] are drawn in turquoise. The two main objectives of the presented design are small memory footprint and only marginal impact on the performance of the delivery of the primary spiking events. Each received primary or secondary event carries the global identifier (`gid`) of the sending neuron. The sparse table indicates at a given `gid` if the sending neuron has at least one (primary or secondary) connection on the local machine. That given, the sparse table provides a pointer to the attached connection containers. A spike event to be delivered is passed to all primary connections that are found below the pointer. Even though

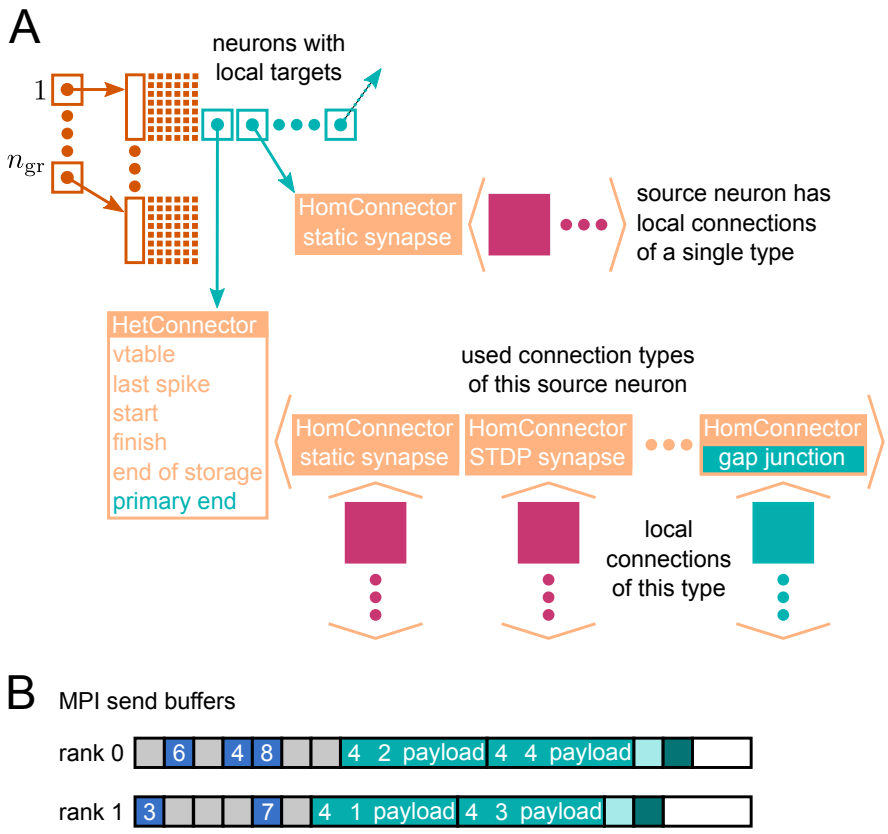

Figure 5.6: **Data structures for the representation of gap junctions.** Turquoise elements indicate necessary changes to the fundamental data structures with respect to the 4g simulation kernel of NEST (cf. [112]). **(A)** Thread-local connection infrastructure. For all neurons a sparse table (dark orange) encodes whether at least one thread-local target is present or not. If a neuron has local targets, the sparse table stores a pointer (turquoise square with arrow) to a connection container (light orange data structure), where the least significant bits of this pointer encode whether gap junctions are present or not. The container is either a `HomConnector` or a `HetConnector` depending on whether the neuron has only one or more than one type of local connection. A `HomConnector` directly stores the connection objects, whereas a `HetConnector` stores a vector of `HomConnectors`, one per connection type. The `HomConnectors` for spiking connections come first in the `vector` and the member `primary_end` is the number of spiking connection types in the `vector`. **(B)** MPI send buffers accumulating outgoing events in the scheduler. Toy example for a particular communication interval with two MPI processes, where rank 0 hosts the neurons with even global IDs (`gids`) and rank 1 hosts the neurons with odd `gids`. Each buffer consists of two parts: the data related to spiking connections (blue boxes) followed by the data related to gap junctions (turquoise boxes).

there are different connection types, such as to distinguish static connections from those exhibiting spike-timing dependent plasticity, all primary connections convey

spike events. The situation is different for the secondary events, because an event containing the interpolation parameters for a gap-junction current should only be delivered to a neuron that expects this information. The latter is indicated by an existing gap-junction connection from the sending neuron. The identification between secondary events and the corresponding connection is achieved by a unique identifier that is assigned to each secondary synapse type and its corresponding event type upon registration at the simulation kernel.

The adaptive data structure presented in [112] in the limit of large machines collapses along the dimension of synapse type, realized by the homogeneous connector `HomConnector` in Fig. 5.6A. As a consequence, if a given source neuron only has a single target connection on a given machine or several connections of the same type, the additional infrastructure provided by the `HetConnector` (the linear searchable array of different connection types, the member `primary_end`) is not available. In this case, we need a separate mechanism to decide whether or not a received primary or secondary event is to be delivered to a particular target. For reasons of performance, this decision is done in two steps. In the first check, in the case of a primary (spiking) event, we determine if the target neuron has at least one primary connection; correspondingly, for a secondary event if it has at least one secondary connection. To perform this test as early as possible and without the use of either an additional data member or the need to parse the full connection structure below the pointer, we make use of redundant information in the pointer contained in the sparse table. As pointer addresses are aligned to at least double word boundaries, their two lowest significant bits are always zero. We use the lowest significant bit to indicate whether or not the sending neuron has at least one primary connection, the second lowest significant bit to indicate the existence of at least one secondary connection. This first test can hence be done directly after retrieval of the pointer from the sparse table, which only happens if the neuron has at least one connection of any type, be it primary or secondary. To access the pointed to data structure we disregard the two lowest significant bits. The second decision depends on the container being homogeneous (containing only connections of a unique type) or heterogeneous. Delivering a primary event to a homogeneous or heterogeneous connector does not require any additional checks. The delivery of a secondary event to a homogeneous connector requires the comparison of the secondary event identifier to match the identifier of the stored connections which by definition are all the same. For heterogeneous connectors this requires a linear search in the list of secondary connections to find the connection type that matches the secondary event type. This is typically affordable, as each neuron typically only has a few different incoming secondary connection types, if any at all. To find the initial point of the linear search in the list of targets shown in Fig. 5.6A, the heterogeneous connector `HetConnector` holds the index `primary_end` of the last primary connection type.

5.1.3 Communication infrastructure

The payload events, introduced in Subsec. 5.1.2 represent the path by which neurons exchange arbitrary data. In contrast to primary spiking events that only carry the `gid` of the sending neuron and the time stamp of event occurrence, a payload event transports additional information. We use this concept as an abstraction layer to the underlying MPI-based [124] data exchange. To this end, payload events support serialization of their contents into the MPI send buffers and de-serialization of these events from the MPI receive buffer. For reasons of performance, these buffers are homogeneous arrays of unsigned integers. Upon serialization, the payload event first writes out its unique type identifier, followed by its length as measured in multiples of unsigned integers, followed by its payload. Upon reception, this process can without ambiguities be inverted, as the unique type identifier of the payload events allows the identification of the corresponding event type on the receiver side. Syntactically we use streaming operators (`GapJEvent::operator>>(vector<unsigned int>::forward_iterator &)`, and the corresponding `operator<<`) to encapsulate the serialization and de-serialization, which requires static type casting. To avoid the duplication of data, the `GapJEvent` does not hold the array of coefficients for interpolation directly, but rather holds iterators for the beginning and end of the corresponding coefficient arrays. On the sending side, these iterators point to the interpolation coefficients that are stored in the neuron. Upon collation of the send buffers (in function `gather_events` of the scheduler, see Algorithm 5.3), these coefficients are directly copied once from their respective neurons to the MPI send buffer. On the receiving side, the same iterators point to the positions in the receive buffer that hold the corresponding coefficients. The iterator class internally represents the positions as `vector<unsigned int>::iterator` to allow fast copy into the MPI send buffers by standard algorithms (`std::copy`). In addition, for convenience on the side of the neuron, it defines an iterator interface (with functionality to increment and dereferenciation) for the represented data type, in case of the `GapJEvent` for `double`.

Algorithm 5.1 shows the use of the `GapJEvent` in the update loop of the neuron. After the interpolation coefficients have been collected during a preliminary update, the coefficient array is passed to a newly created `GapJEvent`, which internally sets the iterators accordingly, and is then sent to the network via the method `send_secondary` that registers the event in the scheduler. When the above mentioned streaming operators are employed, upon registration, secondary events are directly serialized into a separate buffer of unsigned integers for each thread. Prior to the communication step, the final send buffer is collocated by the call of the function `gather_events` (see Algorithm 5.3) by first collecting the spiking events grouped according to the time slices in which they occurred, as illustrated in Fig. 5.6B. The buffers may not be completely filled, as they are adapted as

Algorithm 5.1: **Update function of a neuron model supporting gap junctions.** The `update_neuron` function propagates the state of the neuron from t_{left} to t_{right}. The state of the neuron at time t is the vector $y = (y_V, \ldots, y_{\text{syn}})$, where $y_V(t)$ denotes the membrane potential that includes the gap current $I_{\text{gap}}(t)$. $I_{\text{gap}}(t)$ is given by (5.5) and depends on the chosen method of interpolation. The analytic solution in line 13 symbolically represents the integrator for the differential system. y_{syn} represents the component of the synaptic input current, which is affected by incoming synaptic impulses in line 15. The function returns `true` if the stopping criterion is satisfied.

```
1  bool update_neuron(t_left, t_right, bool prelim_update)
2
3    // neuron is in state y(t_left)
4
5    done = true
6    n_steps = (t_right - t_left)/h
7    n_coeff = (n_order + 1) · n_steps
8    new_coefficients[i] = 0   ∀i ∈ {0,...,n_coeff-1}
9
10   for k ∈ {1,...,n_steps}:
11       // solve differential equation y'(t) = f(y(t))
12       // using ḡ and g̃_i (see Alg. 5.2 for definition of g̃_i) according to (5.5)
13       y(t_left + kh) = y(t_left + (k − 1)h) + ∫_{t_left+(k−1)h}^{t_left+kh} f(y(s)) ds
14
15       y_syn = y_syn + input_buffer[t_lag] // set new synaptic input current
16
17       if (not prelim_update):
18         // check for threshold and refractoriness
19         if not refractory:
20           if y_V > Θ:
21             emit spike and set neuron refractory for time τ_r
22         else:
23           decrease refractory counter
24       else: // preliminary update
25         // collect coefficients of membrane potential interpolation
26         for s ∈ {0,...,n_order}:
27           new_coefficients[(k − 1) · (n_order + 1) + s] = a_{i,s} // as shown in (Tab. 5.1)
28         // check if stopping criterion is violated
29         if(| V_last[k − 1] − y_V | ≥ prelim_tol):
30           done = false
31         V_last[k − 1] = y_V
32
33   if (not prelim_update):
34     for k ∈ {1,...,n_steps}:
35       new_coefficients[(k − 1) · (n_order + 1)] = y_V(t_right) // constant extrapolation
36       V_last[k − 1] = 0 // reset V_last
37
38   // send interpolation coefficients to network as gap event
39   GapJEvent ge(new_coefficients);
40   send_secondary(ge);
41
42   // reset data for interpolation
43   ḡ = 0
44   g̃_i = 0  ∀i ∈ {0,...,n_coeff-1}
45
46   return done
```

Algorithm 5.2: `handle` **function algorithm**. The `handle` function receives the `GapJEvents` and collects the gap weights and interpolation coefficients according to (5.5). In contrast to (5.5) the vector $\tilde{g}$ holds the values for all time steps s within one iteration interval, instead of just for one fixed time step, i.e. $\tilde{g}_{s+(k-1)\cdot(n_{\text{order}}+1)} := g^*_{k,i,s}$

```
1 handle(GapJEvent e)
2
3   // n_coeff given as in (Alg. 5.1)
4
5   ḡ = ḡ + e.g
6   for i ∈ {0,...,n_coeff − 1}:
7     g̃_i = g̃_i + e.g · e.coefficients[i]
```

soon as more data needs to be transmitted but are not reduced in size in the case of less data. The number of time slices $d_{\min}/h$ per communication interval of duration $d_{\min}$ is, however, fixed and the end of each time slice is marked by a special identifier (shown as gray square). Consequently, the receiving side knows when all spiking events have been read. Therefore, in direct succession to the spiking data, the secondary events buffer for each thread is appended to the send buffer. A reserved identifier, `invalid_id,` marks the end of the secondary events, followed by a boolean value, indicating whether or not the desired accuracy has been achieved in the current iteration step, as explained in Subsec. 5.1.4.

5.1.4 Iterative neuronal updates

On the scheduler level, the iterative simulation of a time interval is implemented in lines 9-20 of Algorithm 5.3. Instead of just once, the update function of the involved neurons (Algorithm 5.1) is called several times to perform so-called *preliminary updates* before the final update is done and the time of the simulation is advanced. The precise number of preliminary calls to the update function is determined by the iteration control, as introduced in Subsec. 5.1.1. Each neuron returns to the scheduler if its solution achieved the desired accuracy. The scheduler summarizes the feedback and sends the information to the other MPI processes. The result over all MPI processes is then returned to the scheduler to determine if a further preliminary iteration is needed.

The discrimination between the preliminary updates and the final update is necessary, since during a preliminary update the neuron will not issue any spiking events, as shown in Algorithm 5.1. The incoming spiking events in each iteration are hence the same. On the other hand, only within a preliminary update a neuron will send secondary events conveying the interpolation of its membrane

Algorithm 5.3: **Pseudo code of the `simulate` function in the scheduler.** Lines 9-20 show the additional code due to the new preliminary updates. An additional boolean value passed to the update function of a neuron distinguishes a preliminary update (`true`) from the final update (`false`). The `gather_events()` function builds the send buffer including the boolean value of the variable `done`, that indicates whether or not additional iterations are needed, and performs the MPI communication. The `deliver_events()` function distributes the received events locally and returns `true` only if all MPI processes indicated that the desired accuracy was achieved. The function `advance_time()` updates the values of t_{left} and t_{right} to the boundaries of the next time interval.

```
1  simulate()
2
3    [...]
4
5    //t_left , t_right  given
6
7    deliver_events()
8
9    //preliminary updates
10   for i ∈ {1,...,max_num_prelim_iterations}:
11       // done indicates if iteration has converged
12       // or more preliminary updates are needed
13       done = true
14       for all neuron that need prelim_update:
15         done = update_neuron(t_left , t_right , true) and done
16
17       gather_events(done)
18       done = deliver_events()
19       if(done):
20         break
21
22   //final update
23   for all neurons:
24     update_neuron(t_left , t_right , false)
25   gather_events(true);
26   advance_time()
27
28   [...]
```

potential to its peers. The final, non-preliminary update conveys the extrapolation of the membrane potential to the other neurons, which will be used in the first iteration of the next time step. Fig. 5.7 shows the realization of the iterative update process for two neurons with special focus on the communication of the interpolation coefficients.

The first computation of the time step is calculated with a constant extrapolation

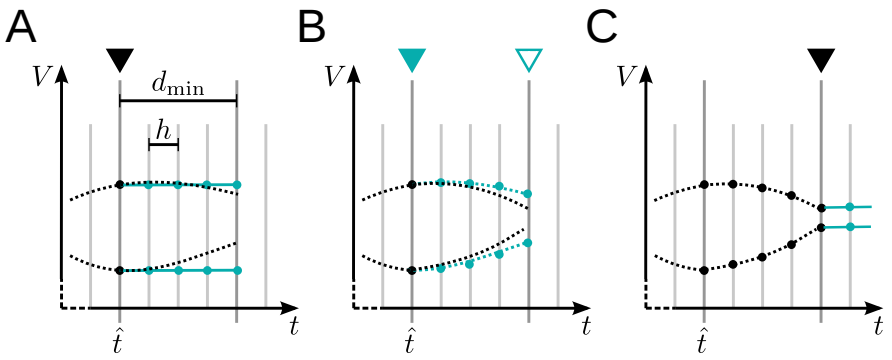

Figure 5.7: **Iterative neuronal updates.** Communication of spikes and gap-junction related data is carried out in steps of $d_{\min}$ (long gray lines), which denote the minimum synaptic transmission delay in the network. Within each communication interval, neurons update their dynamics in steps of h (shorter light gray lines); here $d_{\min} = 4h$ at time $\hat{t}$. Turquoise curves show the approximation of the membrane potential, which is used by the connected neuron to compute the solution in the current interval. **(A)** First iteration with constant approximation for the membrane potential of the connected neuron. At the end, a new approximation of the just computed membrane potential is passed to the connected neurons. **(B)** Further iteration with the approximation of the membrane potential from last iteration. This part is the actual iteration process, which can be done multiple times. **(C)** After the final iteration a constant extrapolation for the next time step is communicated.

of the membrane potential of the connected neurons. In every further iteration of the same time interval the interpolation generated with the last iteration is used. Accordingly, the interpolation of the current membrane potential is computed during preliminary iterations, while for the final iteration a constant extrapolation is send to the scheduler. The interpolation coefficients are computed as described in Subsec. 5.1.1 and saved in an array. The same applies for the receiving side (Algorithm 5.2), where the coefficients from the incoming connections are accumulated as described in Subsec. 5.1.1.

The neuron update function shown in Algorithm 5.1 has a boolean parameter to distinguish if the current call is a preliminary or the final update. The implementation can be used with both communication strategies, since communication in every time step is only a special case ($d_{\min} = h$) that can be induced manually by creating an unused primary connection with delay h and therefore does not require further adaptation to the code.

5.2 Neuron model

The neuron model used for our study is a point-neuron model with Hodgkin-Huxley dynamics. The model was introduced by Mancilla et al. [118] to investigate the synchronization of electrically coupled pairs of inhibitory interneurons in the neocortex. We prefer this model over a leaky integrate-and-fire model (see e.g. [65], their Chapter 4.1) because the former naturally includes the time course of an action potential, while this is a point-event in the latter. The membrane potential of the model fulfills the ordinary differential equation

$$V' = \frac{-I_{\text{ionic}}(V, m, h, n, p) + I_{\text{ex}} + I_{\text{in}} + I_{\text{gap}}}{C_m}$$

with

$$\begin{aligned} I_{\text{ionic}} &= g_{\text{Na}} m^3 h (V - V_{\text{Na}}) \\ &\quad + (g_{\text{Kv3}} p^2 + g_{\text{Kv1}} n^4)(V - V_{\text{K}}) \\ &\quad + g_{\text{leak}} (V - V_{\text{leak}}) \\ I_{\text{gap}} &= \sum_j g_{ij} (V_j - V)\,. \end{aligned}$$

The channel dynamics is given by

$$\begin{aligned} m' &= \alpha_m (1 - m) - m\beta_m \\ h' &= \alpha_h (1 - h) - h\beta_h \\ n' &= \alpha_n (1 - n) - n\beta_n \\ p' &= \alpha_p (1 - p) - p\beta_p. \end{aligned}$$

A spike is transmitted to the network if a non-refractory neuron passes the threshold. The time of the spike is defined as the first grid point after the time where V has reached its maximum value. Without restricting the generality of the results, we model synaptic events as currents described by alpha functions (see [153], their Sec. 3.1.2). The total excitatory and inhibitory synaptic input currents read

$$I_{\text{ex}}(t) = \sum_{k=1}^{n_{\text{ex}}} J_k \cdot H\left(t - t_{\text{ex},k}\right) \cdot e\, \frac{t - t_{\text{ex},k}}{\tau_{\text{ex}}} \cdot \exp\left(\frac{-\left(t - t_{\text{ex},k}\right)}{\tau_{\text{ex}}}\right) \tag{5.6}$$

$$I_{\text{in}}(t) = \sum_{k=1}^{n_{\text{in}}} J_k \cdot H\left(t - t_{\text{in},k}\right) \cdot e\, \frac{t - t_{\text{in},k}}{\tau_{\text{in}}} \cdot \exp\left(\frac{-\left(t - t_{\text{in},k}\right)}{\tau_{\text{in}}}\right), \tag{5.7}$$

where the J_k denotes the synaptic weight and $H(x) = \begin{cases} 0 & x \leq 0 \\ 1 & x \geq 0 \end{cases}$ denotes the Heaviside step function. The times at which spikes arrive at the neuron are indicated by $\{t_{\mathrm{ex},k}\}_{k=1,\ldots,n_{\mathrm{ex}}}$ and $\{t_{\mathrm{in},k}\}_{k=1,\ldots,n_{\mathrm{in}}}$. In the time interval between two subsequent spike times $t_{\mathrm{ex},k}$ and $t_{\mathrm{ex},k+1}$ the current (5.6) (and analogously also (5.7)) can be expressed as the solution of two additional ordinary differential equations

$$I'_{\mathrm{ex}}(t) = I_{\mathrm{ex2}}(t) - \frac{I_{\mathrm{ex}}(t)}{\tau_{\mathrm{ex}}} \tag{5.8}$$

$$I'_{\mathrm{ex2}}(t) = -\frac{I_{\mathrm{ex2}}(t)}{\tau_{\mathrm{ex}}} \tag{5.9}$$

with initial conditions

$$\begin{aligned} I_{\mathrm{ex}}(t_{\mathrm{ex},k}) &= I^{\mathrm{old}}_{\mathrm{ex}}(t_{\mathrm{ex},k}) \\ I_{\mathrm{ex2}}(t_{\mathrm{ex},k}) &= I^{\mathrm{old}}_{\mathrm{ex2}}(t_{\mathrm{ex},k}) + J_k \cdot \frac{e}{\tau_{\mathrm{ex}}} \end{aligned}$$

where $I^{\mathrm{old}}_{\mathrm{ex}}(t_{\mathrm{ex},k})$ and $I^{\mathrm{old}}_{\mathrm{ex2}}(t_{\mathrm{ex},k})$ denote the solutions from the previous interval between $t_{\mathrm{ex},k-1}$ and $t_{\mathrm{ex},k}$. For the first interval we define $I^{\mathrm{old}}_{\mathrm{ex}}(t_{\mathrm{ex},1}) = I^{\mathrm{old}}_{\mathrm{ex2}}(t_{\mathrm{ex},1}) = 0$.

The equivalence of the formulations (5.6) and (5.8) can be verified by looking at the analytical solutions of (5.8) and (5.9)

$$\begin{aligned} I_{\mathrm{ex}}(t) &= \left(I^{\mathrm{old}}_{\mathrm{ex}}(t_{\mathrm{ex},k}) + I^{\mathrm{old}}_{\mathrm{ex2}}(t_{\mathrm{ex},k})(t - t_{\mathrm{ex},k}) + J_k \cdot e\, \frac{t - t_{\mathrm{ex},k}}{\tau_{\mathrm{ex}}} \right) \exp\left(\frac{-(t - t_{\mathrm{ex},k})}{\tau_{\mathrm{ex}}} \right) \\ I_{\mathrm{ex2}}(t) &= \left(I^{\mathrm{old}}_{\mathrm{ex2}}(t_{\mathrm{ex},k}) + J_k \cdot e \frac{1}{\tau_{\mathrm{ex}}} \right) \exp\left(\frac{-(t - t_{\mathrm{ex},k})}{\tau_{\mathrm{ex}}} \right) . \end{aligned}$$

By starting with the first interval and successively applying the solution from one interval to the subsequent interval one obtains (5.6).

In total, the neuron model therefore consists of a system of nine ODEs. Further information on the parameters and settings can be found in [118]. The NEST implementation `hh_psc_alpha_gap` of the model uses the GNU Scientific Library implementation (`gsl_odeiv_step_rkf45`) of the Runge-Kutta-Fehlberg 4(5) method (see Table 2.1) with adaptive step-size control (`gsl_odeiv_control_y_new`) to advance the state of an individual neuron by the interval h. Thus, the solver may use finer steps to cover the interval according to the demands of the dynamics. The accuracy parameter for the absolute predicted error made in each interval h is chosen as 10^{-6}; the parameter for the relative predicted error is not used and set to 0. Therefore, there is no use in choosing the `prelim_tol` parameter of the waveform relaxation below the former value.

5.3 User interface

The NEST `Connect` routine enables neuroscientists to express a partial network structure through the connections between two sets of neurons. One dictionary specifies the connection rule and the rule-specific parameters, a second the dynamics of the interaction. The implementation as of version 2.10.0 [110] accepts various connection rules from simple ones, like `all_to_all` and `one_to_one`, to random connections between the sets, such as `fixed_indegree` and `fixed_total_number`. Chemical synapses, the original research domain of NEST, imply a directed interaction. Therefore, the `Connect` routine is designed to specify directed graphs. Gap junctions, however, mediate a bidirectional interaction. Simulation code for spiking neuronal networks exploits the directedness of chemical synapses by representing synapses only on the compute node where the postsynaptic neuron resides. This enables network creation to be organized as an ideally parallelized activity without communication between nodes (see Subsec. 2.3.1). In this framework, gap junctions need partial representations on the postsynaptic as well as on the presynaptic side to mediate the bidirectional interaction on the undirected subgraph. Hence, in order to connect two neurons through a gap junction, connections in both directions need to be created.

Script 5.4 shows various ways to connect two previously created neurons with a single gap junction in NEST 2.10.0 and 2.12.0 in the PyNEST [51] syntax. In order to prevent the creation of incomplete or non-symmetric gap junctions, NEST 2.12.0 introduces a `make_symmetric` flag with default value false to the connection algorithm. This flag provides the option to create both connections with a single call to `Connect`, as in the second variant of Script 5.4, by creating the reverse connection internally. With the introduction of the flag, the creation of gap junctions is restricted to `one_to_one` connections with true `make_symmetric` flag and to `all_to_all` connections with equal source and target populations and default or scalar parameters. Therefore, the first variant cannot be used in NEST 2.12.0 or later. The third variant, starting at line 14, generalizes to all-to-all connected networks of an arbitrary number of neurons, independent of whether a network of chemical synapses or gap junctions is desired. Self connections are excluded by setting the `autapses` flag to false. All three variants ideally and automatically parallelize, relying on the NEST implementation of `Connect`.

The restriction on the creation of gap junctions is mostly motivated by more complex random networks, where we need to take special care. Let us consider an example where the total number of gap junctions in a given volume of cortical tissue is known. These gap junctions are randomly distributed over all possible pairs of neurons in the volume without any further constraints. In particular, a neuron does not have a gap junction with itself, but a given pair of neurons may be coupled by more than one gap junction.

Script 5.4: **Various ways to create a gap junction between two neurons.** Two connections are required; one for each direction. Here and in the following scripts we use the syntax of the PyNEST interface [51] of the NEST simulation software. By convention, in `Connect(i,j)` the interaction is from i to j; i exerts an influence on j. `Create` returns an n-tuple and `Connect` accepts n-tuples, lists, and arrays of the `numpy` module as arguments for i and j. The third positional argument of `Connect` specifies the connection algorithm; use of a dictionary for the connection algorithm enables the specification of more details. The `make_symmetric` flag introduced with NEST 2.12.0 provides the possibility to create both connections with a single call. In this case the reverse connection is created internally. An autapse is a connection a neuron forms with itself, which is here forbidden in the `all_to_all`-version (line 16). The fourth positional argument specifies the dynamics of the connection; here we just specify the model name `gap_junction`. While the first two alternatives are only meaningful for a single gap junction, the third generalizes to networks with all-to-all connectivity.

```
1 import nest
2
3 n = nest.Create('hh_psc_alpha_gap', 2)
4
5 # only works with NEST 2.10.0
6 nest.Connect([n[0]], [n[1]], 'one_to_one', 'gap_junction')
7 nest.Connect([n[1]], [n[0]], 'one_to_one', 'gap_junction')
8
9 # works with NEST 2.12.0 (or later)
10 nest.Connect([n[0]], [n[1]],
11              {'rule': 'one_to_one', 'make_symmetric': True},
12              'gap_junction')
13
14 # works with both versions
15 nest.Connect(n, n,
16              {'rule': 'all_to_all', 'autapses': False},
17              'gap_junction')
```

Script 5.5 shows a script implementing this network using the `fixed_total_number` algorithm of the `Connect` command. At line 14 the network is created as a directed graph; the interaction is mediated only in one direction. `Connect` efficiently generates this network, instantiating the relevant subgraphs in parallel on all of the compute nodes using parallel random number generators. Therefore on the level of the interpreter executing the script, the actual connectivity is not known. In order to create the complementary directed graph, we need to retrieve the existing connections from the simulation kernel, exchange the roles of pre- and postsynaptic neurons, and in addition create this subnetwork. `GetConnections`, however, only returns the set of incoming connections of the neurons represented

Script 5.5: **Creation of a network with a predetermined total number of gap junctions between randomly chosen pairs of neurons using a predefined connection algorithm.** As a first step (line 14) the random network is created as a directed graph. The second step (lines 17-19) obtains the list of connected neuron pairs from the simulator and reshapes the data to corresponding lists of pre- and postsynaptic neurons. The final step (line 21) adds the transposed connectivity matrix to the network by supplying `Connect` with the lists of the pre- and postsynaptic neurons of the original network in reversed order. The parameters in the script result in a binomially distributed number of gap junctions per neuron with a mean of 60. The script does not work in a distributed simulation as the function `GetConnections` only returns the part of the network represented on the node executing the command: the set of incoming connections of the locally represented neurons. The successful execution of the script is only possible with NEST 2.10.0; starting with NEST 2.12.0 the `Connect` call in line 14 issues an error.

```
1 import nest
2 import numpy as np
3
4 # total number of neurons
5 N = 100
6 # total number of gap junctions
7 K = 3000
8
9 n = nest.Create('hh_psc_alpha_gap', N)
10
11 r = {'rule': 'fixed_total_number', 'N': K, 'autapses': False}
12 g = {'model': 'gap_junction', 'weight': 0.5}
13
14 nest.Connect(n, n, r, g)
15
16 # get source and target of all connections
17 m = np.transpose(
18     nest.GetStatus(nest.GetConnections(n),
19     ['source', 'target']))
20
21 nest.Connect(m[1], m[0], 'one_to_one', g)
```

on the local compute node. The transpose of this subnetwork generally, therefore, has mainly non-local postsynaptic neurons which the `Connect` command ignores. Hence, Script 5.5 does not work in a distributed simulation. This problem occurs for any type of network where the realization is only known to the simulation engine. Users without in-depth knowledge of the connection infrastructure of NEST might therefore create potentially incomplete gap junctions with this or similar

Script 5.6: **Creation of a network with a predetermined total number of gap junctions using an explicit list of random neuron pairs.** Same parameters as in Script 5.5. The random module of the Python Standard Library is used to independently draw K pairs of random samples from the list of all neurons (line 17). The data is in the same line reshaped into two corresponding lists of pre- and postsynaptic neurons. The subsequent `Connect` call uses the syntax of PyNEST as of NEST 2.12.0. The script does work in a distributed simulation.

```
1 import nest
2 import random
3 import numpy as np
4
5 # total number of neurons
6 N = 100
7 # total number of gap junctions
8 K = 3000
9
10 n = nest.Create('hh_psc_alpha_gap', N)
11
12 r = {'rule': 'one_to_one', 'make_symmetric': True}
13 g = {'model': 'gap_junction', 'weight': 0.5}
14
15 random.seed(0)
16
17 m = np.transpose([random.sample(n, 2) for _ in range(K)])
18
19 nest.Connect(m[0], m[1], r, g)
```

scripts in NEST 2.10.0. Starting with NEST 2.12.0, the `Connect` call in line 14 therefore issues an error due to the new restriction mentioned above.

Random connections of gap junctions must therefore be created on the level of the interpreter executing the script before handing them down to the `Connect` command. Script 5.6 illustrates this approach using the `random` module of the Python Standard Library. The drawback of this script is the serialization of the connection procedure in terms of computation time and memory. Each compute node participating in the simulation needs to draw the same full set of random numbers and temporarily represent the total connectivity in variable `m`. In the subsequent call to `Connect`, each compute node only considers those neuron pairs where the postsynaptic neuron is local.

5.4 Numerical results

We employ three network models to study different aspects of the iterative method in comparison with the single-step method. In Subsec. 5.4.2 we investigate the pair of neurons coupled by a gap junction which was already presented in the introduction of this chapter to demonstrate the problems of the single-step method in contrast to the advanced method based on waveform relaxation. For the waveform-relaxation method, we compare the results with linear and cubic interpolation. This approach discloses the general behavior of the methods and provides access to the single-neuron integration error not measurable in recurrent networks with chaotic dynamics (see [81] and [87] for earlier uses of this technique). Subsequently in Subsec. 5.4.3, a network of inhibitory neurons is investigated to demonstrate the simultaneous integration of spiking and gap-junction dynamics and to confirm the accuracy of the iterative method in capturing a parameter value at which a qualitative change in network activity occurs. Finally, in Subsec. 5.4.4 the scalable network model of Kunkel et al. [112] is used to assess the influence of the gap-junction framework on memory consumption and simulation speed in simulations that exclusively use spiking synapses. The performance in simulations with gap junctions is measured with a scaled version of the two neuron test case. Details on the test cases employed, computers and measures of accuracy are summarized in Subsec. 5.4.1. All performance tests in Subsec. 5.4.4 use NEST 2.10.0 [14]. All other numerical tests use a slightly earlier prototype branch of NEST.

5.4.1 Setup

5.4.1.1 Test cases

Test case 1a: pair of neurons coupled by a gap junction The setup consists of two `hh_psc_alpha_gap` neurons i and j connected by a gap junction with weight $g_{ij} = 30.0\,\text{nS}$. Both neurons receive a constant current of 200.0 pA. All other parameters are kept at their default values (see [118]) for both neurons. The minimum delay of spike interaction is set to 1.0 ms.

Test case 1b: scalable network with gap junctions only The test case 1a is extended to N neurons to investigate the scaling of the gap-junction framework. Each neuron is coupled by gap junctions of uniform weight $g = 0.5\,\text{nS}$ to 60 other neurons. For the sake of simplicity neuron i is coupled to the neurons from index $(i-30+N) \mod N$ to $(i+30) \mod N$, where $\mod$ denotes the modulo operator. Thus the 30 prior and the 30 subsequent neurons are coupled if one

considers the neurons aligned in a ring. All other inputs and parameters are the same as in test case 1a.

Test case 2: inhibitory network We investigate a network of 500 neurons of type `hh_psc_alpha_gap` with random initial membrane potentials between $-40\,\mathrm{mV}$ and $-80\,\mathrm{mV}$. Each neuron receives 50 inhibitory synaptic inputs that are randomly selected from all other neurons, each with synaptic weight $J_\mathrm{I} = -50.0$ pA and synaptic delay $d = 1.0\,\mathrm{ms}$. Each neuron receives an excitatory external Poissonian input of 500.0 Hz with synaptic weight $J_\mathrm{E} = 300.0$ pA and the same delay d. In addition, $\frac{60 \cdot 500}{2}$ gap junctions are added randomly to the network, resulting in an average of 60 gap-junction connections per neuron. The weight g of each gap-junction connection is chosen uniformly and will be varied within our tests.

Test case 3: scalable network without gap junctions The setup consists of a balanced random network model [20] of 80% excitatory and 20% inhibitory leaky integrate-and-fire model neurons with alpha-shaped post-synaptic currents studied by Kunkel et al. [112] in a maximum-filling scenario. Both cell types are represented by the NEST implementation `iaf_neuron` with a homogeneous set of parameters. All excitatory-excitatory connections exhibit spike-timing dependent plasticity and all other synapses are static. In [112] the network is used to characterize the differences between the 3rd- and 4th-generation simulation kernel of NEST. We use parameter set 1 of [112] to assess the overhead of the gap-junction framework in simulations where no gap junctions are present.

5.4.1.2 Computers

The numerical results in this chapter are obtained with three different computer systems: a workstation computer, a single shared memory node of a cluster and two distributed-memory supercomputing systems. The workstation is used for the simulation of small networks investigating the accuracy of the methods (test cases 1a and 2), while the simulations on the shared memory cluster and the supercomputers benchmark the scalability of the new approach in terms of run time and memory usage (test cases 1b and 3).

The workstation computer comprises a 4-core Intel(R) Core(TM) i7-4770 processor, which runs at 3.4 GHz and supports simultaneous multithreading with 2 threads per core. 32 GB of random access memory (RAM) are available. The shared memory node of the cluster HAMBACH at the Jülich Research Centre in Germany includes compute nodes with 4 AMD Magny-Cours 12-core Opteron 6174 with 2.2 GHz and 256 GB RAM. For our study the parallelization on both systems is carried out by OpenMP [137].

The employed supercomputers are the JUQUEEN BlueGene/Q [98] at the Jülich Research Centre in Germany and the K computer [126] at the Advanced Institute for Computational Science (AICS) in Kobe, Japan. The former comprises 28,672 nodes, each with a 16-core IBM PowerPC A2 processor which runs at 1.6 GHz. The latter consists of 88,128 nodes, each with an 8-core SPARC64 VIIIfx processor which operates at a clock frequency of 2 GHz.

Both systems support a hybrid simulation scheme: distributed-memory parallel computing with MPI [124] and multithreading (OpenMP) on the processor level. 16 GB RAM are available per compute node for both systems. In the case of JUQUEEN the nodes are connected through a five-dimensional torus interconnect network with a bandwidth of 2 GB/s per link. In case of the K computer, they are connected with the "Tofu" (torus connected full connection) interconnect network, which is a six-dimensional mesh/torus network. The bandwidth per link is 5 GB/s.

In this chapter, all supercomputer benchmarks were run with 8 OpenMP threads per compute node and the pool allocator (see [112] for details). These are the same settings as in the former work of Kunkel et al. [112], which allow us to compare with previous results (see Subsec. 5.4.4).

For the remainder of this chapter, simulation results on different hardware systems are distinguished in the figures by color: shades of green for workstations and shared memory clusters, shades of blue for the JUQUEEN BlueGene/Q and shades of red for the K computer.

5.4.1.3 Measures of accuracy

Different measures are used to determine the accuracy of the solution. The initial two measures compare the membrane potential $V(t)$ to a known reference solution $\widehat{V}(t)$. To demonstrate that the integration method can qualitatively change the network dynamics, we also use further measures which characterize the emergent properties of the network such as the firing rate and synchrony.

Error in membrane potential time course First, we employ the well known root mean square error (RMSE)

$$\epsilon \;=\; \sqrt{\frac{1}{T}\int_0^T (\widehat{V}(t) - V(t))^2 dt}, \tag{5.10}$$

which measures the deviation of the solution $V(t)$ for the membrane potential from a reference solution $\widehat{V}(t)$ on a time interval $[0, T]$.

Since the solution is unknown in continuous time, a discrete approximation with linear interpolation between the grid points is used as in [87]. This first order approximation with n grid points at times $t_1, ..., t_n$ with $\Delta V_k = \widehat{V}(t_k) - V(t_k)$ and $\Delta t_k = t_k - t_{k-1}$ can be determined as

$$\epsilon \approx \sqrt{\frac{1}{3T} \sum_{k=1}^{n-1} \Delta t_{k+1} (\Delta V_k^2 + \Delta V_{k+1}^2 + \Delta V_k \Delta V_{k+1})}. \tag{5.11}$$

In contrast to the mean relative error measure

$$l_2 = \frac{\sqrt{\int_0^T (\widehat{V}(t) - V(t))^2 dt}}{\sqrt{\int_0^T \widehat{V}(t)^2 dt}} = \frac{\epsilon}{\sqrt{\frac{1}{T} \int_0^T \widehat{V}(t)^2 dt}},$$

which was employed to determine the error in the membrane potential time course in [153], the RMSE calculates the mean absolute error. We decided to use the latter as our error measure ϵ, since the behavior of the membrane potential in our test cases is well known, which makes the absolute error a more descriptive measure.

Temporal displacement Secondly, we introduce a measure for the shift δ between $V(t)$ and the reference solution $\widehat{V}(t)$. This measure determines the relative time shift τ between the two signals that minimizes the RMSE

$$\delta = \underset{0 \leq \tau \leq \tau^*}{\operatorname{argmin}} \sqrt{\frac{1}{T} \int_0^T (\widehat{V}(t) - V(t+\tau))^2 dt}. \tag{5.12}$$

Of course, this error measure is only reasonable if the RMSE is indeed caused by a shift. In addition, periodic signals can lead to misleading results if, for example, there is a shift of exactly one period. Nevertheless, the shift is a descriptive measure if the neurons under consideration match the required conditions. For practical usage, we employ the same discretization as for the RMSE and calculate $V(t_k + \tau)$ through linear interpolation between the grid points.

Network synchrony For a network of N neurons with membrane potentials $V_1, ..., V_N$, we determine the degree of synchrony of the network state as in [78] and [130] by the temporal fluctuations $\sigma_{V_{\text{av}}}$ of the average membrane potential

$$V_{\text{av}}(t) \;=\; \frac{1}{N}\sum_{i=1}^{N} V_i(t)$$

normalized by the average temporal fluctuation σ_{V_i} of the single cells in the population. The resulting measure $\chi(N)$ reads

$$\chi(N) \;=\; \sqrt{\frac{\sigma^2_{V_{\text{av}}}}{\frac{1}{N}\sum_{i=1}^{N}\sigma^2_{V_i}}} \tag{5.13}$$

and covers the interval from 0 to 1, where 1 denotes a fully synchronized network and 0 denotes the asynchronous state. The variance $\sigma^2_{V_{\text{av}}}$ (and analogously $\sigma^2_{V_i}$) can be estimated as

$$\sigma^2_{V_{\text{av}}} \;\approx\; \frac{1}{T}\int_0^T \left[V_{\text{av}}(t)\right]^2 dt - \left[\frac{1}{T}\int_0^T V_{\text{av}}(t)\,dt\right]^2.$$

For our calculations, the occurring integrals are approximated by the trapezoidal rule

$$\frac{1}{T}\int_0^T x(t)dt \;\doteq\; \frac{1}{n-1}\left(\frac{x(t_1)+x(t_n)}{2} + \sum_{k=2}^{n-1} x(t_k)\right).$$

Average spike rate For a given spike train $S(t)$ with spikes at time $t_1, \ldots, t_n$, the spike count function $\eta(t)$ counting the number of spikes that have occurred up to and including time t can be written as

$$\eta(t) \;=\; \sum_{k=1}^{n} H(t - t_k)$$

where again $H(x)$ denotes the Heaviside step function. We determine the spike rate ν in the interval $(0, T]$ as

$$\nu \;=\; \frac{\eta(T)}{T},$$

and denote the average spiking rate of a network of N neurons as

$$\bar{\nu} \;=\; \frac{1}{N}\sum_{i=1}^{N}\nu_i.$$

This estimate of the spike rate ν is consistent with the assumption that $n(t)$ is a homogeneous Poisson process with fixed intensity, for which we try to estimate the intensity by the given realization (see [99], their Chapter 19).

5.4.2 Pair of gap-junction coupled neurons

We employ a pair of gap-junction coupled neurons with identical parameters and constant input current (test case 1a) to investigate the accuracy on the single neuron level. Since both neurons behave exactly the same, their membrane voltages are identical and consequently $I_{\text{gap}} = 0$ holds at all times. Therefore the results of a consistent gap-junction implementation should be exactly the same as for two uncoupled neurons with the same properties. In this setting, the results of the uncoupled pair of neurons can be used as a reference solution to determine the quality of the investigated integration methods. The employed gap weight g of 30.0 nS represents the typical total coupling of a single neuron with the remainder of the network: the natural weight of a single gap junction is much smaller, but each neuron is connected to dozens of other neurons. The test case exposes how the numerical methods operate on networks of synchronized neurons coupled by gap junctions. In the absence of any chemical synapses, the minimum delay of spike interaction is set to 1.0 ms in order to obtain realistic results for the integration scheme that only communicates when spike times need to be exchanged.

Fig. 5.8A shows the functionality of the iterative method by measuring the error ϵ in the membrane potential for different numbers of iterations. The RMSE decreases with every iteration until it converges to some plateau error. The plateau error depends on the used interpolation order and is independent of the employed communication strategy. Its origin will be discussed later in Fig. 5.10. As expected, a faster convergence is reached with the h-step communication, while the communication in intervals of the minimal delay takes a few more iterations. The lower panel (Fig. 5.8B) shows the mean number of iterations when the same simulation is run with the iteration control described in section Subsec. 5.1.1. The number of needed iterations is mostly independent of the step size h and the used interpolation order, but differs by about four iterations for the different communication strategies.

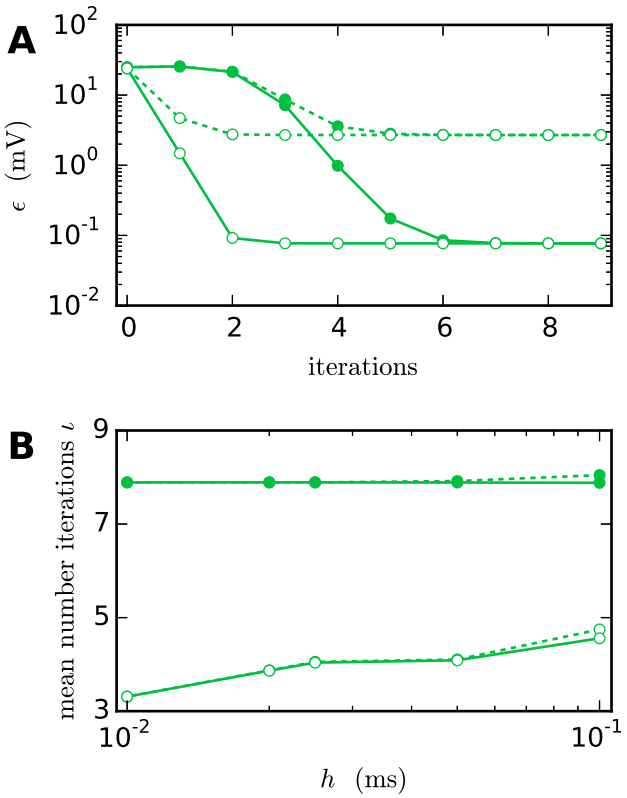

Figure 5.8: **Integration error as a function of the number of iterations.** Solid curves indicate cubic interpolation, dashed curves linear interpolation. Filled circles show results for the communication interval of NEST communication, open circles show the results for communication in every time step h. Color indicates the hardware system; in this and all subsequent figures shades of green represent workstations (here) or shared memory cluster node. The RMSE ϵ of the membrane potential was measured over 1 s of biological time. The step size h was chosen as 0.05 ms leading to $r = \frac{1.0}{0.05} = 20$ time steps within one minimal delay communication interval. **(A)** RMSE for different numbers of iterations. **(B)** Mean number of iterations when using the iteration control with default settings (`prelim_tol` chosen as 10^{-4} and a maximum of 15 iterations, which was not reached for any simulation interval).

In simulations with distributed memory the total number of communications is an important quantity, as each communication is associated with a considerable latency. If we denote with ι_h and $\iota_{d_{\min}}$ the mean number of iterations with the corresponding strategy and define as $r = \frac{d_{\min}}{h}$ the number of time steps per minimal delay interval, the total numbers of communications in each step C_h and after each minimum delay $C_{d_{\min}}$ relate to each other as

$$C_h \approx \frac{\iota_h}{\iota_{d_{\min}}} \cdot r \cdot C_{d_{\min}}. \tag{5.14}$$

As the coupling strength of the test case relates to the total coupling of a single

neuron, the simulation provides a realistic estimate for the number of iterations needed within larger network simulations. For this given estimate, C_h exceeds $C_{d_{\min}}$ for $r \geq 3$. Since $r = 10$ or $r = 20$ are more likely for an average simulation, we have $C_h \gg C_{d_{\min}}$. Therefore communication after each minimum delay is beneficial for the reduction of the total number of communications despite the faster convergence of the h-step communication strategy.

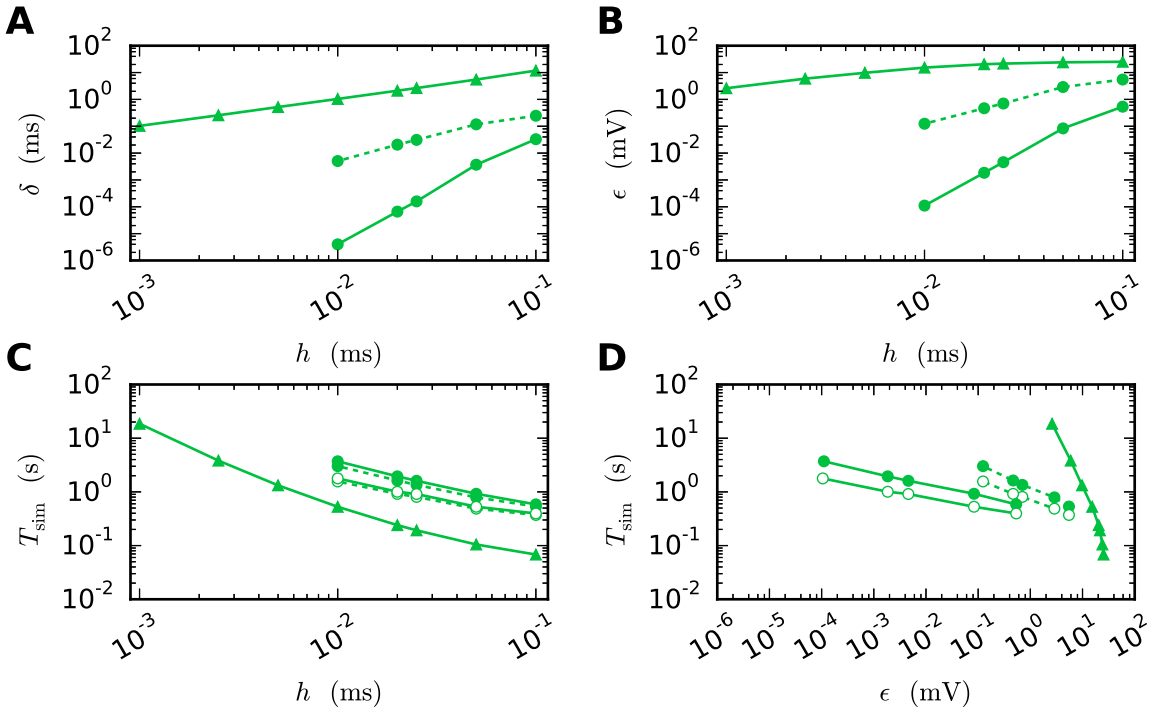

Figure 5.9: **Efficiency of a two-neuron simulation.** Triangles show results with the single-step method, while circles indicate results obtained by the iterative method. Again solid curves indicate cubic interpolation and dashed curves were obtained with linear interpolation. The used communication scheme is indicated by open (h-step communication) and filled symbols (communication in intervals of the minimal delay). The iteration control was used with `prelim_tol` chosen as 10^{-6}. **(A)** Shift of the spike times after 1 s of biological time plotted against used step size h. **(B)** RMSE ϵ measured over 1 s of biological time plotted against the step size h. **(C)** Simulation time of the different approaches for 1 s of biological time. **(D)** Simulation time versus RMSE ϵ of the corresponding simulation.

Fig. 5.9 compares the results of the iterative method with the results of the single step methods in terms of accuracy and simulation time. Panel B measures the error ϵ of both methods for different step sizes h. For any given step size h the RMSE of the iterative method is much smaller than the RMSE of the single-step approach, which does not even reach a satisfying accuracy for step

size $h = 0.001\,\text{ms}$. Within the iterative method, cubic interpolation leads to a higher accuracy. For the cubic interpolation the RMSE reduces from 0.52 for $h = 0.1$ to $1.01 \cdot 10^{-4}$ for $h = 0.01$. Fig. 5.9A shows that the error relates to a shift in comparison to the reference solution. This shift can be reduced up to $10^{-6}\,\text{ms}$ for the iterative method with cubic interpolation and step size 0.01 ms. For a given step size h and leaving accuracy considerations aside, the single step method is the fastest implementation for any given step size, since no additional iterations are needed to compute the results. The iterative approach with linear interpolation saves some time in comparison to the version with cubic interpolation, since less interpolation data needs to be computed and communicated. For this simple test case, h-step communication outperforms the communication strategy in intervals of the minimal delay by a factor of 1.5, due to the very low amount of communicated data and because the communication in the employed shared memory system is fast compared to the computation. Further simulation time results for simulations on supercomputers are presented in Subsec. 5.4.4. Fig. 5.9D compares the methods in terms of efficiency. To this purpose we analyze the simulation time as a function of the integration error [130], measured through the RMSE. There are two ways of reading this graph: horizontally, one can find the most accurate method for a given simulation time. Vertically one can find the fastest method for a desired accuracy. The results show that the iterative method delivers better results in shorter time than the single step method. Also, the additional effort of the cubic simulation pays off, since the method computes more accurate results in the same simulation time and reaches an accuracy which cannot be reached with the linear interpolation.

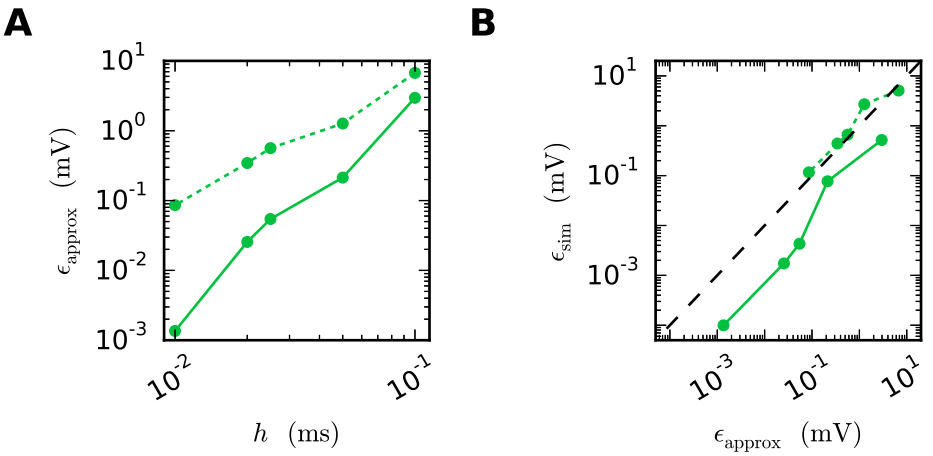

Figure 5.10: **Effect of membrane potential interpolation on network error. (A)** RMSE ϵ_{approx} of linear (dashed curves) and cubic (solid curves) interpolation for the action potential shown in Fig. 5.4 as a function of the computation step size h. **(B)** Integration error for the two-neuron network (Fig. 5.9) as a function of the interpolation error shown in (A).

We observe from Fig. 5.8 and Fig. 5.9 that the error of the iterative method converges to a certain plateau whose magnitude depends on the step size. Fig. 5.10 shows that the approximation of the membrane potential is the reason for this inaccuracy. The left panel shows the error when approximating the reference spike shape in Fig. 5.4 with different step sizes. The right panel compares this error to the NEST RMSE when using the corresponding interpolation and same step size h. It turns out that the errors are basically the same, as indicated by the dashed line. The approximation error of the cubic interpolation is slightly higher than the mean simulation error, since the approximation of a spike is the most difficult part of the interpolation. The membrane potential between two spikes can be approximated almost perfectly with a cubic interpolation, although the spike shape still deviates from cubic behavior.

The contour plots in Fig. 5.11 show the influence of gap weight, step size and desired accuracy `prelim_tol` on the error ϵ and the number of iterations. Panel A demonstrates that the order of magnitude of the parameter `prelim_tol` only relates to that of the RMSE if the step size h is sufficiently small. For larger step sizes the accuracy is limited by the interpolation error. Panel B shows that the error is also influenced by the gap weight, i.e. by the coupling strength in the network. Especially for step sizes $h > 0.05\,\text{ms}$, a weaker coupling between the neurons results in a higher accuracy. Fig. 5.11C and D show that the mean number of iterations is dependent on both the parameter `prelim_tol` and the gap weight g for both communication strategies. As already indicated by Fig. 5.8, the number of iterations is lower for h-step communication, regardless of the exact choice of both parameters.

5.4.3 Network with combined dynamics of chemical synapses and gap junctions

The results in Subsec. 5.4.2 show the functionality of the iterative approach in purely gap-junction coupled networks. This section now investigates whether the numerical method based on waveform relaxation also captures the global network dynamics correctly, when both chemical synapses and gap junctions are involved, which is another important aspect for the employed method. In order to do so, we turn to population measures like the spike rate and synchrony in the network and study a network with a phase transition. Capturing the correct parameter value at which the transition occurs is a good indication that not only the single neuron error is low but also the global error. This idea and the network setting have a history in [79], [130], [36] and [78]. The employed network (test case 2) consists of 500 neurons with external excitatory Poissonian input, which are coupled by inhibitory synapses and gap junctions with uniform gap weight g. Without the gap junctions (i.e. for $g = 0\,\text{nS}$) the network shows an asynchronous

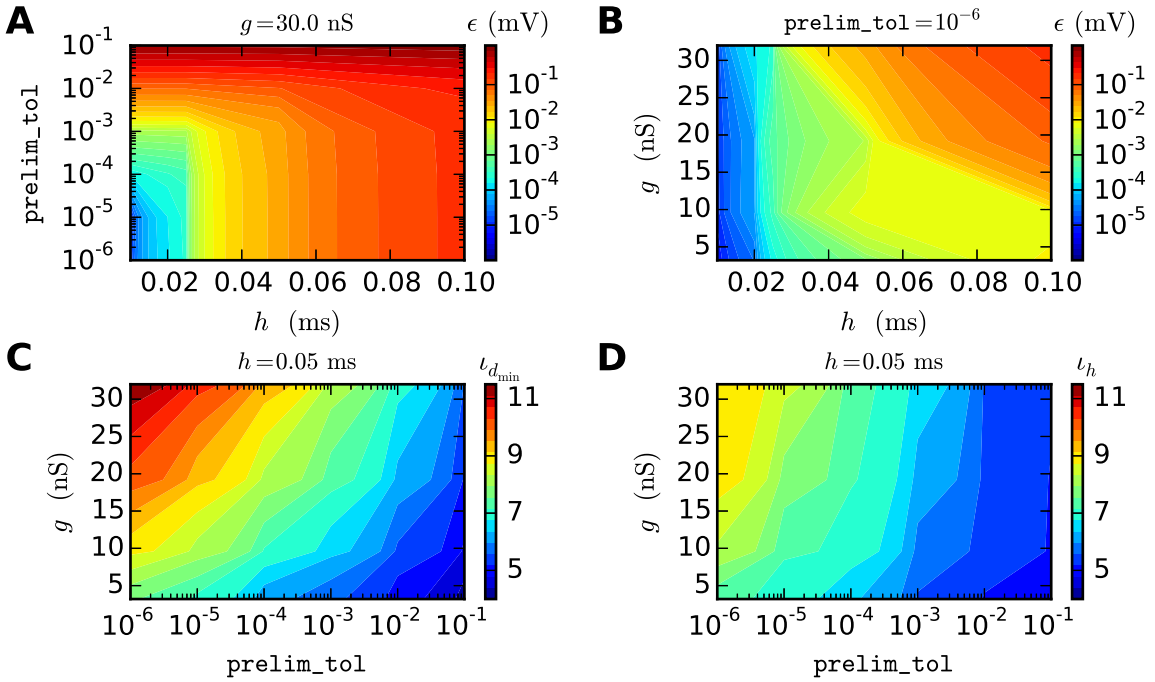

Figure 5.11: **Influence of gap weight, step size and desired accuracy on the error ϵ and the number of iterations.** The data for the contour plot is obtained by several simulations of test case 1a with various gap weights g, different desired accuracies `prelim_tol` and different step sizes h. All simulations run for 100 ms of biological time and use cubic interpolation. Panels (A), (B) and (C) employ communication in intervals of the minimal delay and panel (D) employs h-step communication. **(A)** RMSE in dependency of step size h and desired accuracy `prelim_tol`. The gap weight is fixed at $g = 30.0$ nS. **(B)** RMSE in dependency of step size h and gap weight g. The desired accuracy `prelim_tol` is fixed at 10^{-6}. **(C)** The mean number of iterations $\iota_{d_{\min}}$ in dependency of desired accuracy `prelim_tol` and gap weight g. The step size is fixed at $h = 0.05$ ms. **(D)** Same setup as in panel (C) but with mean iterations measured for the case of h-step communication.

irregular state [21] that is caused by the external excitatory Poissonian drive being balanced by the inhibitory feedback within the network. The network is expected to synchronize with increasing g. A qualitatively similar synchronization has been observed previously [36]. In this setup it is natural to regard the gap weight g as the bifurcation parameter.

Fig. 5.12 shows the spiking behavior of the employed network for different choices of the gap weight g. For a lower gap weight $g = 0.3$ nS the network remains in an asynchronous state. In panel B ($g = 0.54$ nS) the network switches randomly between the asynchronous and synchronous state, while for the highest gap weight

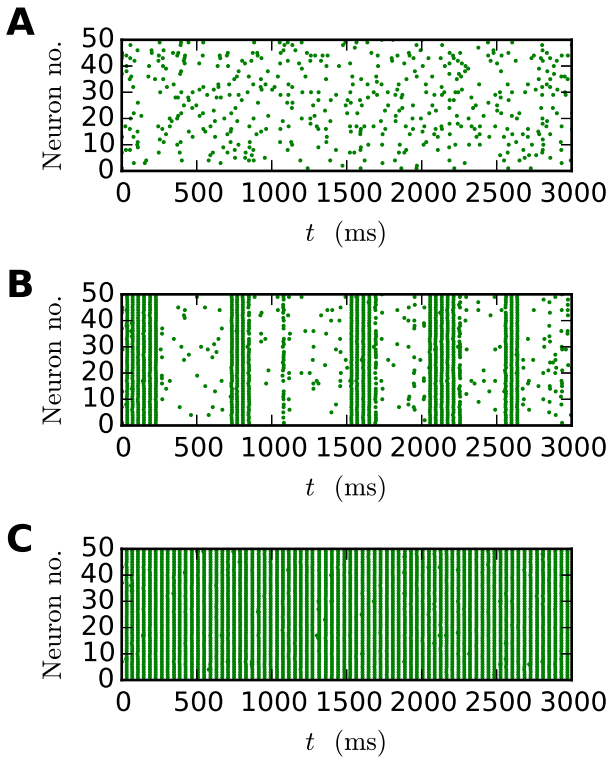

Figure 5.12: **Spike patterns for different gap weights.** The panels show the spike times of the first 50 neurons of the inhibitory network (test case 2) over 3 s of biological time for $J_I = -25$ pA. All results were obtained with the iterative method with cubic interpolation and step size 0.05 ms. **(A)** gap weight $g = 0.3$ nS **(B)** gap weight $g = 0.54$ nS **(C)** gap weight $g = 0.7$ nS

$g = 0.7$ nS, a stable synchronous state is reached. The exact transition between these two states as a function of the gap weight and choice of integration method is visualized in Fig. 5.13. To overcome statistical fluctuations caused by the random transitions between the asynchronous and the synchronous state, which can be observed in Fig. 5.12B, the system needs to be simulated for a prolonged time. The transition is investigated for two different choices of the synaptic weight of the inhibitory synapses to demonstrate the influence of the chemical synapses on the location of the transition. The shift of the transition point between both choices of J_I guarantees the influence of the chemical synapses on the global network dynamics, which is needed in order to show the correctness of the new iterative method for networks with chemical synapses and gap junctions.

As expected, an increase of the gap weight leads to a higher network synchrony, which also influences the spike rate. For the two choices of J_I Fig. 5.13 shows differences in the gap weight at which the network turns from the partly synchronized state to the almost fully synchronized state. In order to demonstrate the

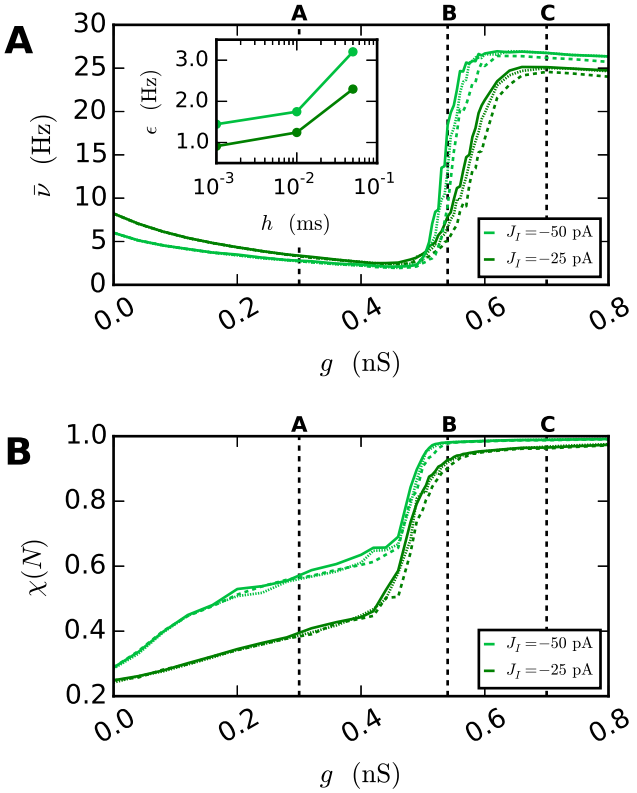

Figure 5.13: **Network behavior depending on the gap weight g.** The average spike rate $\bar{\nu}$ **(A)** and the synchrony χ (5.13) of the neurons in the network **(B)**, depending on the gap weight. The results for the iterative method with cubic interpolation are shown as solid curves (step size 0.05 ms) and for the single-step method with dashed (step size 0.05 ms) and dotted (step size 0.001 ms) curves. Two different synaptic amplitudes $J_I = -50$ pA and $J_I = -25$ pA were used, as indicated by the figure legend. The `prelim_tol` was chosen as 10^{-5} and the maximum number of iterations was not used as a stopping criterion. The simulation duration was 100 s ($J_I = -25$ pA), respectively 180 s ($J_I = -50$ pA) of biological time. The inset of panel A shows the difference between the results of the iterative method (step size 0.05 ms) and the results of the single-step method for different step sizes h measured by the RMSE. The dashed vertical lines correspond to the panels of Fig. 5.12.

correctness of the new iterative method over the single-step method, the latter is simulated for different step sizes h. The inset of Fig. 5.13A shows the difference of the spike rate (measured through the RMSE) between the two methods, depending on the step size. It demonstrates that the solution of the single-step method converges to the solution of the iterative method. In agreement with the results presented in Subsec. 5.4.2, the convergence is slow: even for the step size $h = 0.001$ ms the difference is still apparent.

Regardless of the parameter value at which the transition occurs, the inaccuracy of the single-step method is also noticeable in the spike rate for higher gap weights ($g > 0.6$ nS), as the influence of the employed method increases with the gap weight used. The lower spike rate of the single-step method is an immediate consequence of the previously seen shift. The shift goes along with a longer distance between two spikes, which leads to the observed lower spike rate.

5.4.4 Performance of the gap-junction framework in NEST

The design of the framework for gap junctions in NEST, as described in Sec. 5.1, is guided by the goal to neither impair code maintainability or impose penalties on run time or memory usage for simulations that exclusively use chemical synapses. The first requirement is addressed by the design choice to tightly integrate the novel framework with the existing connection and communication infrastructure of NEST instead of developing an independent pathway for gap-junction related data. Therefore, we are interested in the performance of i) simulations exclusively using chemical synapses and ii) simulations including gap junctions.

We employ the balanced random network model from [20] (test case 3) to investigate simulations exclusively using chemical synapses. We measure the deviation in simulation time and memory usage between the last release without the gap-junction framework (NEST 2.8.0, [52]) and NEST 2.10.0. Although NEST 2.10.0 also contains other changes and new features, like a framework for structural plasticity, the most time- and memory-sensitive changes are due to the gap-junction framework. Fig. 5.14 shows results for the network in a maximum-filling scenario, where for a given machine size N_{VP} we simulate the largest possible network that completely fills the memory of the machine. Although the simulation scenario is maximum filling, we were able to simulate the same network size as before, as the increase in memory usage is within the safety margin of our maximum-filling procedure (see [112] for details on the procedure). Measured in percentage of the prior memory usage (Fig. 5.14B), the consumption increases by 0.2 to 1.5 percent depending on the number of virtual processes N_{VP}. The behavior on JUQUEEN and the K computer is almost identical. The run time of the simulation increases up to 4.0 percent for simulations with a low number of virtual processes, with an average of 1.1 and 0.7 percent on JUQUEEN and the K computer respectively. The simulation times on the K computer show slightly higher fluctuations, although the measurements are averaged over three runs on both supercomputers. The small increase of memory usage is caused by changes to the thread-local connection infrastructure and the communication buffer described in Subsec. 5.1.2 and Subsec. 5.1.3. In case of primary events only (no use of gap junctions) the only extra data member is `primary_end`, which only affects the connection container called `HetConnector`. As the `HetConnector` is only instantiated if there

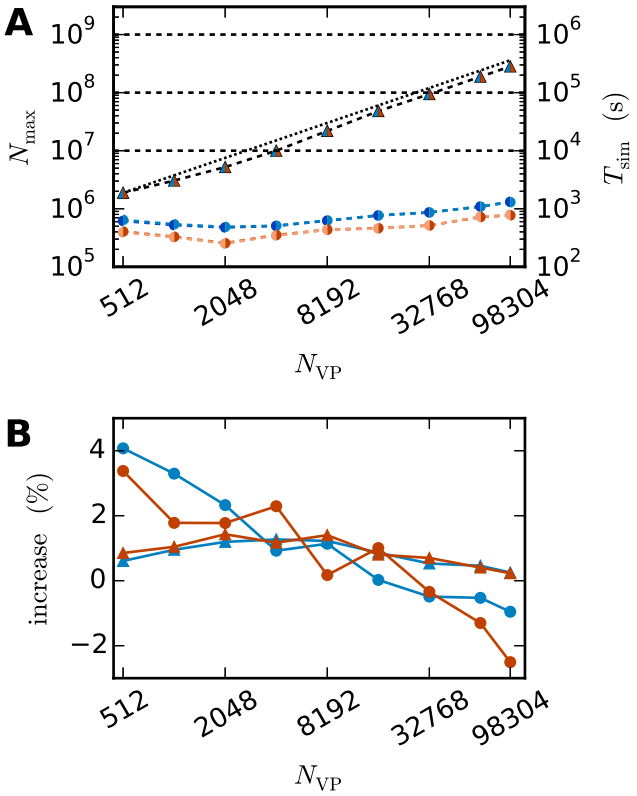

Figure 5.14: **Overhead of the gap-junction framework for network with only chemical synapses.** N_{VP} denotes the overall number of processes used in line with our distribution strategy, as described in Subsec. 5.4.1.2. In this and all subsequent figures, shades of blue indicate the JUQUEEN supercomputer and shades of red show the results for the K computer. **(A)** Triangles show the maximum network size that can be simulated in the absence of gap junctions (test case 3). Circles show the corresponding wall-clock time required to simulate the network for 1 s of biological time. Left semicircles indicate the results with NEST 2.8.0 without the gap-junction framework and right semicircles are obtained with the framework included (NEST 2.10.0). **(B)** Increase of time (circles) and memory consumption (triangles) of NEST 2.10.0 in percent as compared to NEST 2.8.0.

are two or more synapse types targeting neurons on a given machine and having the same source neuron, this additional data member is irrelevant in the limit of large machines (sparse limit), where practically all connections are stored in `HomConnectors`; the latter containers only hold connections of identical types and do not have the additional data member `primary_end`. The small increase of the run time is due to an additional check for existence of secondary connections, which has to be done during the delivery of the events. The check is done directly after retrieving the pointer address from the sparse table and does not require additional memory, as this information is encoded in redundant bits of the pointer

address itself (see Subsec. 5.1.2 for details). The reduced increase of the run time at higher numbers of virtual processes N_{VP} is due to the prolonged simulation time, as some part of the overhead is caused by the initialization in the beginning of the simulation.

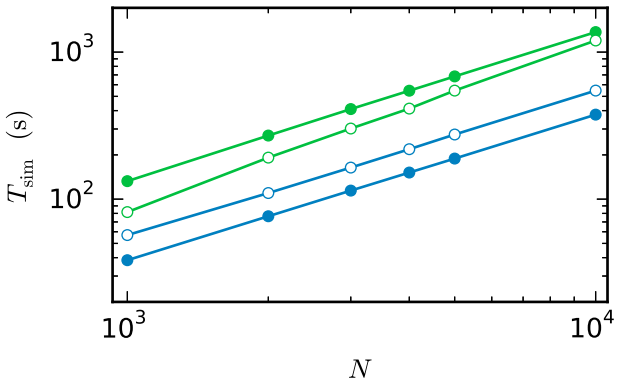

Figure 5.15: **Comparison of simulation times on different systems.** Simulation of the scaled version of the pair of neurons (test case 1b) with different network sizes N. Open symbols show the results for communication in every step ($\mathcal{T} = h$) while filled symbols show the results for the original NEST communication scheme ($\mathcal{T} = d_{\mathrm{min}}$). The simulations on the workstation (green) are executed with 8 virtual processes (8 threads). The JUQUEEN simulations (blue) use 128 virtual processes (16 MPI processes a 8 threads). 500 ms of biological time are simulated with step size $h = 0.05\,\mathrm{ms}$.

Next we turn to simulations with gap junctions. The benchmarks use a scaled version of the network simulating a pair of neurons (test case 1b), where each neuron is coupled to 60 other neurons by gap junctions. The number of neurons performing the computation and the amount of communicated data increase with N. We keep the conductance of a neuron accumulated over all gap junctions the same as in the case of the network comprised of a single pair (test case 1a). As a consequence, the computations carried out by the integrator of each individual neuron are the same, and hence its dynamics is independent of N. Thus, the performance of the gap-junction framework can be measured in a setting with fixed single neuron dynamics despite the presence of additional neurons.

Fig. 5.15 compares the run time of both communication strategies on JUQUEEN with their performance on workstations. On the workstation, the h-step communication performs better due to the smaller number of iterations per interval and the fast communication through shared memory. On JUQUEEN, however, the communication in d_{min} steps outperforms communication in every step. As discussed in Subsec. 5.4.2, the total number of communications (5.14) of the h-step communication strategy C_h exceeds $C_{d_{\mathrm{min}}}$. Due to the latency of the communication in a system with distributed memory, the original NEST communication

strategy performs better on JUQUEEN despite the comparatively small number of 16 MPI processes.

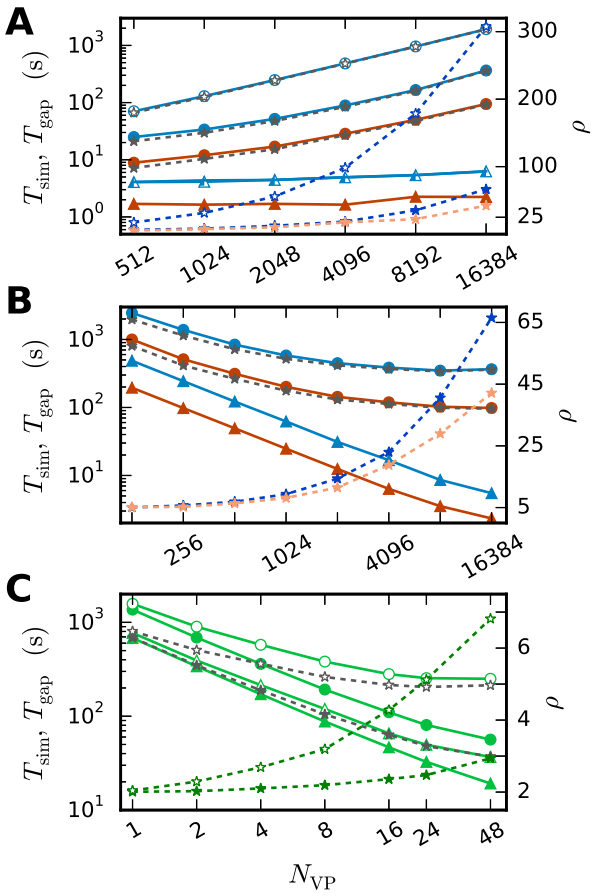

Figure 5.16: **Costs of the gap-junction dynamics.** Open symbols show the results with h-step communication ($\mathcal{T} = h$), while filled symbols show the results with the original NEST communication scheme ($\mathcal{T} = d_{\min}$, here $d_{\min} = 1\,\text{ms}$). The solid curves with triangles indicate the simulation time T_{sim} in the absence of gap junctions. The corresponding darker curves with asterisks show the ratio ρ of T_{sim} with and without gap junctions, while gray curves with asterisks show the difference T_{gap} of both simulation times. Simulations represent 50 ms of biological time for panel A and B and 100 ms for panel C at a step size of $h = 0.05\,\text{ms}$. All simulations use only a single iteration per time interval. **(A)** Weak scaling of test case 1b on JUQUEEN and the K computer with $N = 185 \cdot N_{\text{VP}}$ neurons. **(B)** Strong scaling of test case 1b on JUQUEEN and the K computer with $N = 185 \cdot 16384 = 3,031,040$ neurons. **(C)** Strong scaling of test case 1b run on the shared memory cluster node with $N = 100,000$ neurons.

There are two major differences between simulations with and without gap junctions. First, the iterative waveform-relaxation method requires the repetition of neuronal updates. Since this repetition only multiplies the run time by the number of iterations, it does not affect scalability. Secondly, the approximation of

the membrane potential of each neuron needs to be computed and communicated to its gap-junction coupled partners. As NEST uses `MPI_Allgather` to communicate data between the MPI processes, the receive buffer grows proportionally with the number of neurons. The size of the approximation data $\mathfrak{D}$ of a single neuron depends on the ratio r between the step size h and the minimal delay of the network as

$$\mathfrak{D} = r \cdot (n_{\text{order}} + 1) \cdot 8 \quad \text{(Bytes)}, \tag{5.15}$$

because each time step requires $n_{\text{order}} + 1$ double values to represent the interpolation polynomial between two adjacent time points. All interpolation coefficients are stored as 8 Bytes `double` variables. Consequently, the number of neurons N is a crucial parameter for simulations with gap junctions, as even in a weak-scaling scenario the local memory consumption increases with the global number of neurons. The growth is dominated by the receive buffer and affects both maximum network size and run time. This is not a particular issue of the new iterative method, but rather a general property of the communication by `MPI_Allgather` that also appears in the single-step algorithm.

Fig. 5.16 investigates the slowdown due to gap-junction dynamics. This is done by simulating test case 1b with a single iteration per time interval. The obtained results are then compared to the run time of a simulation without gap junctions but with an otherwise identical setup. This way, the difference of the two run times T_{gap} can be interpreted as the time required for the additional computational load and communication. Fig. 5.16A is a weak-scaling scenario. It demonstrates that the scalability of the method is impaired by the additional communication. Despite the constant number of neurons per virtual process and constant MPI send-buffer size the run time increases substantially. This is due to the increasing total number of neurons, which has an effect on the MPI receive buffer size and thereby on the communication time. Fig. 5.16B studies the same setup in strong scaling with $N \approx 3 \cdot 10^7$ neurons. In this scenario, the receive buffer size is constant, while the size of the send buffer shrinks with increasing number of virtual processes. Here, the additional time required for MPI communication is almost constant. T_{gap} decreases at first and then almost stagnates for more than 2048 virtual processes. The saturation is explained by the additional MPI-communication, which constitutes the major contribution to T_{gap} in this setup. As the simulation without gap junctions uses exactly the same pattern of MPI communication, this is not an issue of latency but an issue of bandwidth. The initial decrease is due to the parallelization of the gap-junction dynamics: the computations on the single-neuron level, like the handling of incoming gap events, the calculation of the interpolation coefficients, and their central storage are parallelized. For both scalings the behavior on JUQUEEN and the K computer is similar. The K computer benefits from the faster processors (2 GHz vs.

1.6 GHz) and the higher bandwidth per link (5 GB/s vs. 2 GB/s), but otherwise shows the same scaling behavior as JUQUEEN.

In conclusion, the additional time required by simulations with gap junctions on both supercomputers is determined by the total number of neurons N. As the increase in run time is dominated by MPI bandwidth it cannot be eliminated by using more virtual processes N_{VP}. Therefore, it is advisable to use as few compute nodes as possible. In this optimal setting, the communication required for gap junctions increases the simulation time of one iteration for a network of $N \approx 3 \cdot 10^7$ neurons by a factor of $\rho = 5.0$. One can multiply this factor ρ from Fig. 5.16 with the average number of iterations ι_h or $\iota_{d_{\min}}$ to receive an estimate of the overall increase in run time.

Fig. 5.16C shows a strong scaling scenario for a smaller network with $N = 100,000$ neurons simulated on a shared memory compute node. This setup differs from the one in panels A and B as the parallelization is implemented by OpenMP and no MPI communication is needed. Here the impact of additional virtual processes on ρ is more moderate. ρ increases from $\rho = 2$ for 2 threads up to $\rho = 3$ for 48 threads in the case where communication takes place in intervals of the minimal delay length. The scalability of NEST is preserved and the time for a single iteration per time interval decreases from 1366 s with 1 thread to 56 s with 48 threads. In contrast to Fig. 5.16B the additional time T_{gap} is not dominated by a constant overhead and decreases due to the parallelization of the gap-junction dynamics. In the case of h-step communication, however, a scaling limit is observed again. The limit is not dominated by the communication between threads but due to the serial component of event delivery in NEST; all threads inspect all incoming events.

In the following part, we employ the memory consumption model of Kunkel et al. [112] to predict the maximum network size which can be simulated with both communication strategies. The model divides the overall memory into three components. $\mathcal{M}_0(\mathfrak{M})$ denotes the base memory usage of the simulator, including external libraries such as MPI, and for the sake of convenience also contains the buffers of the MPI communication. $\mathcal{M}_{\mathrm{n}}(\mathfrak{M}, N)$ is the additional memory usage that accrues when neurons are created, and $\mathcal{M}_{\mathrm{c}}(\mathfrak{M}, \mathfrak{T}, N, \mathfrak{K})$ denotes the additional memory usage that accrues when neurons are connected. The memory consumption per MPI process is thus given by

$$\mathcal{M}(\mathfrak{M}, \mathfrak{T}, N, \mathfrak{K}) = \mathcal{M}_0(\mathfrak{M}) + \mathcal{M}_{\mathrm{n}}(\mathfrak{M}, N) + \mathcal{M}_{\mathrm{c}}(\mathfrak{M}, \mathfrak{T}, N, \mathfrak{K}) \tag{5.16}$$

where $\mathfrak{M}$ denotes the number of compute nodes, $\mathfrak{T}$ the number of threads per node, N the number of neurons and $\mathfrak{K}$ the number of synapses per neuron.

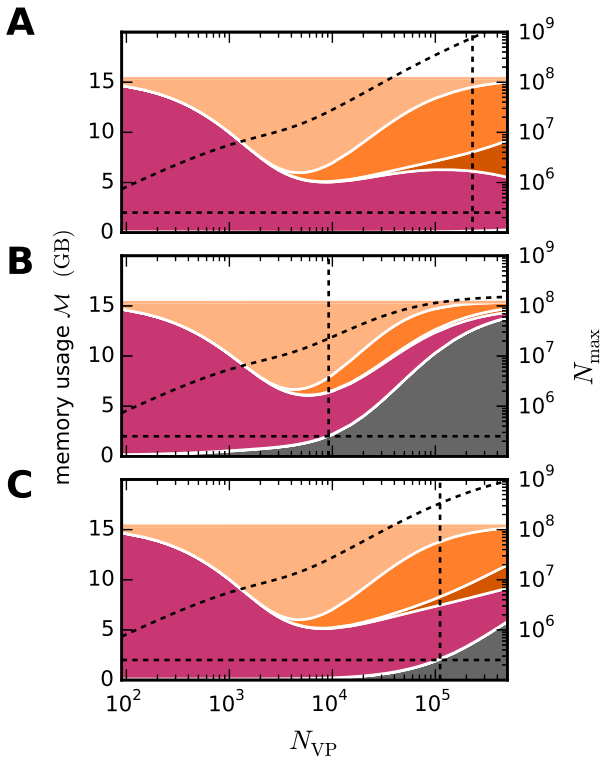

Figure 5.17: **Predicted cumulative memory usage as a function of number of virtual processes for a maximum-filling scaling.** Contributions of different data structure components to total memory usage $\mathcal{M}$ of NEST for test case 3 with a network size that just fits on N_{VP} cores of JUQUEEN. The dashed black curves indicate the corresponding network size $N_{\max}$. Contributions of synapse objects and relevant components of connection infrastructure are shown in pink and shades of orange, respectively. The contributions of the base memory usage, in particular containing the receive buffer, are marked in gray. Other contributions, such as the neuron objects and neuron infrastructure are significantly smaller and hence not visible at this scale. Dark orange: sparse table, orange: intermediate infrastructure containing exactly 1 synapse, light orange: intermediate infrastructure containing more than 1 synapse. The cumulative memory usage is calculated using the memory-consumption model of [112]. The horizontal dashed black line indicates 2 GB limit of a single MPI communication. Vertical lines indicate the largest number of virtual processes possible, due to either the full utilization of JUQUEEN or exceeding a 2 GB communication buffer. **(A)** Test case 3 without gap junctions **(B)** Test case 3 with additional gap junctions between inhibitory neurons (60 gap junctions per neuron). The communication is carried out in intervals of the minimal delay ($d_{\min} = 1.5\,\mathrm{ms}$ and $h = 0.1\,\mathrm{ms}$) **(C)** Same setup as (B) with communication in every time step.

Here we extend the model to include the effect of gap junctions on memory consumption. The memory overhead of neurons with more than one local target $\mathfrak{m}_{\mathrm{c}}^{>1}$ increases by 1 Byte due to the extra data member `primary_end`. The memory

consumption of a connection object of type `gap_junction` $\mathfrak{m}_{\mathrm{c}}^{\mathrm{gap}}$ is the same as for a static synapse $\mathfrak{m}_{\mathrm{c}}^{\mathrm{stat}}$. The memory usage of a single neuron supporting gap junctions $\mathfrak{m}_{\mathrm{n}}^{\mathrm{gap}}$ differs from the usage of another neuron $\mathfrak{m}_n$ by $2\mathfrak{D} + 8r$, as the current interpolation needs to be stored while the new interpolation coefficients are calculated (thus the factor 2), and the values from the last iteration are needed for the iteration control. In addition, the base memory usage $\mathcal{M}_0(\mathfrak{M}, N_{\mathrm{gap}})$ is dependent on the number of neurons supporting gap junctions N_{gap}, as it increases by $(N_{\mathrm{gap}} + \frac{N_{\mathrm{gap}}}{\mathfrak{M}}) \cdot \mathfrak{D}$ due to the increases of the send and receive buffer.

Fig. 5.17 shows the contributions of gap junctions to the memory consumption under maximum filling for the network model introduced by Brunel [20]. This is test case 3 with the addition of gap junctions between inhibitory neurons. We use this network model here to simplify the comparison to existing benchmarks [112]. Dynamically, the network model as is would not be able to support gap-junction coupling, as the leaky integrate-and-fire model `(iaf_neuron)` employed in this test case does not produce the shape of the action potential. Hence, the interactions across gap junctions exerted by the large and positive membrane potential excursions are missing; see below for the range of neuron models available in the literature for the study of networks with gap junctions. Nevertheless the test case provides a good estimate for the additional memory usage caused by gap junctions, as the memory usages of neuron models `iaf_neuron` and `hh_psc_alpha` do not differ significantly relative to the total amount of memory consumed by chemical synapses and gap junctions. The figure shows that with increasing number of virtual processes N_{VP}, the base memory component containing the MPI communication buffers becomes the dominant consumer. This is particularly apparent for communication in intervals of the minimal delay, as the volume of data communicated at once is r times higher than for the h-step communication. As communication in NEST is carried out in a single `MPI_Allgather` call, there is another relevant limit to the MPI buffer size. According to the MPI standard [124] the `recvcount` parameter counting the elements in the receive buffer is an integer value. This limits the largest possible receive buffer size to 2 GB for machines with 32 bit integer values. Therefore, the maximum network size decreases from $8 \cdot 10^8$ for the case without gap junctions to $2 \cdot 10^7$ for the communication in intervals of the minimal delay and to $3.5 \cdot 10^8$ for the h-step communication. This is, however, not a limitation of the iterative numerical method described in this thesis, but a consequence of the overall communication scheme of the NEST simulation software.

5.5 Discussion

It may seem odd to discuss the integration of neuronal networks coupled by gap junctions in the context of a simulation code for which the major application area is large networks of highly simplified spiking neuron models. In these models the occurrence of an action potential is often abstracted to a threshold operation and the shape of the action potential is neglected because it has no influence on network dynamics. This changes, however, in the presence of gap junctions as the gap current depends on the difference of the membrane potentials, mediating an instantaneous coupling. The sign of the coupling depends on the time courses of the two membrane potentials. The positive exertion during an action potential of one cell creates an excitatory drive, while the fast after-hyperpolarization immediately following the depolarization has the opposite effect. Components of the after-hyperpolarization with a longer time constant but low amplitude still have, on average, an inhibitory effect. The network level dynamics may depend not only on the integral effects of action potential and after-hyperpolarization, but also on the time course of the interaction pattern.

Nevertheless, simulation codes designed to faithfully represent the architecture of neuronal networks typically contain a phenomenological step in their neuron models that extracts a spike time to trigger synaptic events and organize communication between the computational nodes participating in a simulation. This is independent of whether the neuron model employed is an integrate-and-fire type model or a model based on the morphological reconstruction of thousands of compartments and a detailed representation of the spiking dynamics generated by the interplay of voltage-gated ion channels. Therefore, these simulation codes have to solve the common problem of how to combine the exchange of spikes as point events with gap-junction coupling without losing the performance capability which originally motivated the design. The algorithms and data structures from our implementation in the NEST simulator and the accompanying analysis can be transferred to other simulation codes and also to digital neuromorphic hardware like SpiNNaker [59].

The developed iterative method based on waveform relaxation guarantees a high accuracy for network simulations with gap junctions, regardless of the coupling strength. For networks with relatively weak coupling, a sufficient accuracy can also be achieved using the less time consuming single-step method (Fig. 5.13, before the transition phase). Here the additional expenses of the iterative method are, however, low, due to the integrated iteration control. For networks with sufficiently strong coupling the single-step approach causes a shift in the membrane potentials time course (Fig. 5.3). This temporal shift reduces with the step size of the simulation. In practice, however, a researcher may not be able to judge whether the coupling strength in the network model under considera-

tion is weak enough to achieve sufficiently accurate results with the single-step method. Furthermore, when the step size of the single-step approach is reduced to improve accuracy, the iterative method eventually achieves a better tradeoff between computation time and accuracy (Fig. 5.9D).

To facilitate generalization, we use as an example a neuron model with Hodgkin-Huxley dynamics that intrinsically generates an action-potential time course and was used to study the synchronization dynamics of networks with gap junctions [118]. However, the literature contains a range of point-neuron models suitable for interaction by gap junctions. These include alternative models of similar complexity, such as the Wang-Buzáki model [184] that represents a fast spiking interneuron in hippocampus or cortex, but also reduced models that combine analytical tractability with the salient features of action-potential generation, including a brief after-hyperpolarization, such as the absolute integrate-and-fire model ([100], reviewed in [36]), and finally, the exponential integrate-and-fire model [58] and the quadratic integrate-and-fire model [77] representing an intermediate level of complexity. Although the gap-junction current I_{gap} in this study is implemented within the employed neuron model as $g_{ij}\ (V_i - V_j)$, the novel gap-junction framework in general is able to process any form of gap junction that depends only on the involved neurons states and parameters. The necessary changes are limited to an adaptation of the neuron model and the creation of a new connection type in the hierarchy of data structures (Fig. 5.6) to distinguish the different representations of gap junctions. A prominent example for a more complex gap-junction model are voltage dependent gap junctions [143]. For these, the optimized summation of coefficients (5.5) is no longer possible, resulting in a higher storage load of the single neuron. Due to the modular structure of NEST, researchers interested in understanding network dynamics can start with a detailed, possibly multi-compartmental, neuron model and then change to a more abstract and analytically tractable model while investigating the network dynamics for qualitative changes. Different parts of the network may also be described at a different level of detail.

Prior to our work simulation studies with gap junctions have only been carried out with network sizes up to a few hundred neurons and extremely simplified topologies such as all-to-all connectivity and only one or two cell types. For example the seminal work of Pfeuty et al. [146] investigates a network of 1600 neurons with random gap-junction coupling and an average of 10 gap junctions per neuron. The numerical integration of the entire network is done using a second-order Runge-Kutta scheme and a fixed step size of 0.01 ms. Although the approach yields accurate results it is not parallelizable and therefore not applicable to substantially larger networks. These initial studies were useful to understand fundamental properties of networks with gap junctions and to verify that the new analytical tools developed are accurate. For neocortical networks, however, this

size constitutes a dramatic downscaling. The number of chemical synapses per neuron is of order 10,000 and the average connection probability within a volume of a cubic millimeter where a neuron can in principle contact any other neuron is about 0.1. Thus, the minimal network size where both of these parameters can simultaneously be realized is 100,000; two orders of magnitudes larger than the networks studied up to now. The need to study neuronal networks at their natural scale has recently gained urgency by the finding that when downscaled, first order measures, such as spike rate, can often be well preserved but already second order measures, like the correlation coefficient of the spike times of two neurons, are generally not preservable [179]. We assume that the primary reason for the present restriction of network size found in the literature is simply due to the technical difficulties in efficiently simulating larger systems, and the absence of a commonly available simulation code providing such capabilities for point-neuron models. The NEURON simulation software [29] provides two multiprocessor solvers for gap junctions between electrical compartments. One incorporates the here presented single-step method using the modified Euler integration scheme into the first order backward Euler integration scheme for tree cables [93]. The other uses the Sundials variable time step, variable order, ODE solver (CVODE, [33]) to solve the global set of equations for all cells. The latter is generally too costly for large spiking network simulations, as the arrival of every spike constitutes a new initial value problem. Nevertheless, the solver was successfully used in the simulation of a gap-junction coupled heart cell network [30]. The novel technology presented here overcomes this limitation; now networks with gap junctions can be studied at full scale. As above in the discussion of the complexity of neuron models, this does not mean that researchers have to carry out all simulations at full scale. Simulation results should simply be checked with full-scale simulations to verify that they do not occur as an artifact of downscaling. The same is true for analytical results derived in the limit of infinite network size. Researchers should verify that the results hold for networks of natural size.

The additional run time costs due to the inclusion of gap junctions in an existing network simulation depend on the number of neurons in the model, as well as the kind of parallelization and the coupling strength of the gap junctions in combination with the desired accuracy. Let us look at a model of the cortical microcircuit as published by Potjans and Diesmann [149] and available as open source (`www.opensourcebrain.org/projects/potjansdiesmann2014`). The model represents a surface area of about 1 square millimeter of cortex and has approximately the same number of neurons as the test case studied in Fig. 5.16C. The model can easily be simulated on a single node of the compute cluster used in the present study, but is time expensive because of the short synaptic delays. The simulation takes 128 seconds with a single thread for 100 ms of biological time and about 16 seconds using 16 threads (data not shown). Fig. 5.16C shows that, for a network of the same size with gap junctions, using the same computational

resources, and the same communication interval of 0.05 ms, the time consumed by the gap-junction dynamics in a single iteration is reduced from 804 seconds to about 215 seconds. The single threaded simulations show that the additional numerical computations required for gap junctions increase run time by a factor of $\frac{128+804}{128} \cdot \iota_h \approx 7.3 \cdot \iota_h$. In a realistic application, the number of iterations ι_h required to reach the desired accuracy goal is below 9 (Fig. 5.11D) and does not affect the scaling because no additional communication is done. At 16 threads, the scaling of the network with gap junctions reaches saturation due to the large number of communication steps. Communication with a minimal delay of 1 ms reduces T_{gap} of a single iteration by a factor of 3.4 to 63 seconds and restores scaling. This corresponds to the reduction of the simulation time by a factor of 11 relative to the single threaded simulation. In conclusion, networks of the size of 100,000 neurons can comfortably be simulated on a single node of a compute cluster in the presence of gap junctions. The simulation time stays within the same order of magnitude, and with increasing communication interval length the difference diminishes. However, looking at a single iteration in our test case, using an expensive single neuron model, the contribution of gap-junction dynamics to the total run time increases from 51% at a single thread to 77% at 16 threads. For the simple neuron model used in the study of the cortical microcircuit, the initial contribution of gap junctions is already 86% and at 16 threads reaches 93%. Thus, for the latter network model, the additional costs of gap junctions are perceived as more painful.

The component limiting network size is the receive buffer of a computational node, which needs to store on the order of 100 Bytes for each neuron in the network (5.15). With memory in the gigabyte range on a computational node, this limits network size to the order of 10 million (10^7) neurons (see Fig. 5.17). Thus, entire areas of the neocortex can be represented. This is promising because larger networks coupled by long-range connections should be under the influence of chemical synapses only. This opens the possibility to exploit the modularity of neuronal networks in future communication algorithms to reach the human brain scale.

There is still further potential for optimization. In terms of performance, it might be possible to save some computation time by applying a less time consuming solver to the cell equations during the iterations. The benefit of such an approach is, however, limited, as the simulation time is mainly dominated by communication (see Fig. 5.16). In terms of accuracy, given a suitable neuron model, it is possible to combine the gap-junction framework with the capability of NEST to handle spike times independently of the grid spanned by the computation time step [81]. A requirement on the neuron model is that an incoming synaptic impulse causes no step in the membrane potential or in its first derivative, since such discontinuities would preclude an accurate cubic interpolation of the membrane

potential within one time step. The neuron model studied in the manuscript satisfies these requirements, as it features alpha-shaped synaptic currents.

The framework for representing and simulating gap junctions extends the capabilities of a simulation engine for neuronal networks like NEST and widens the domain of applications. However, this comes at the price of a decrease in simulation speed by up to 4.0 percent and an increase of memory consumption of up to 1.5 percent even if no gap junctions are used. This is in contrast to the general strategy of NEST development that a researcher should only pay for features actually needed in a simulation, and that a new release should not be slower or consume more memory than the previous one. The software releases prior to 2.10.0 (2.2.0 and 2.6.0) documented in [83, 112] have concentrated on the reduction of memory consumption and also increased simulation speed. In relation to these advances the overhead of the gap-junction framework is only a minor regression; nevertheless, it constitutes a nuisance future work should strive to overcome.

The limits on the maximal network size that can be studied with the framework presented here arise from the need to communicate approximations of the membrane potential time courses between neurons. As the employed communication scheme uses collective MPI calls, these approximations are sent to all nodes that take part in the simulation, irrespective of whether or not they harbor neurons that require this information. This situation is qualitatively similar to the spike times being collectively communicated. However, there are two quantitative differences, the number of connections per neuron (order 10,000 vs. order 100) and the amount of information communicated (4 Byte per spike / order 100 Bytes per minimum delay). A yet more extreme scenario occurs in simulations of multi-compartment neuron models, where the approximation of the membrane potential time course of a particular compartment is relevant for only a few (order 10 down to 1) other compartments. Future work on the simulation code should assess the potential of targeted communication. Due to the low number of connections and their locality, directed communication will be particularly beneficial for gap-junction coupling.

Neuroscience is still challenged by the heterogeneity of the constituents of the neuronal tissue. We hope that the progress reported here adds gap junctions as another type of brick to the Lego kit of the computational neuroscientist. The network sizes reachable with the technology described in this chapter, combined with the supercomputers available today, enable researchers to investigate the functional role of gap junctions in the context of an anatomically accurate circuitry.

Chapter 6

Application in computational neuroscience II: Including rate models in a spiking neural network simulator

This chapter presents our second major use case for waveform-relaxation methods in computational neuroscience. Here we describe how to include rate models in a spiking neural network simulator.

Spiking neural network simulators are based on *bottom-up* approaches that are motivated by the microscopic dynamics of individual neurons. Biophysically grounded spiking neuron models that simulate the time points of action potentials can explain a variety of salient features of microscopic neural activity observed *in vivo*, such as spike-train irregularity [7, 170, 173, 180], membrane-potential fluctuations [46], asynchronous firing [20, 50, 141, 151], correlations in neural activity [62, 85, 136], self-sustained activity [107, 134], rate distributions across neurons [69, 105, 157] and across laminar populations [149], as well as resting state activity [42]. Furthermore, in population-density approaches, statistical descriptions of neuronal populations neglect the identities of individual neurons and describe the dynamics of homogeneous populations in terms of probability densities (reviewed e.g. in [44]). These approaches capture the time-dependent population activity enabling the investigation of phenomena like desynchronization [39] and computational properties of cortical circuits [28].

In contrast, functionally inspired so-called *top-down* approaches typically describe neurons or neuronal populations in terms of continuous variables, e.g. firing rates [89, 163]. Rate-based models originate from the seminal works by Wilson and Cowan [186] and Amari [5] and were introduced as a coarse-grained description of the overall activity of large-scale neuronal networks. Being amenable to

mathematical analysis and exhibiting rich dynamics such as multistability, oscillations, traveling waves, and spatial patterns (see e.g. [155]), rate-based models have fostered progress in the understanding of memory, sensory and motor processes including visuospatial working memory, decision making, perceptual rivalry, geometric visual hallucination patterns, ocular dominance and orientation selectivity, spatial navigation, and movement preparation (reviewed in [16, 35, 102]). On the brain scale, rate models have been used to study resting-state activity [43] and hierarchies of time scales [31]. Ideas from functional network models have further inspired the field of artificial neuronal networks in the domain of engineering [82].

Simulation of rate-based models goes back to the works by Grossberg [70], McClelland and Rumelhart [122], Feldman and Ballard [56], and the PDP group [158]. Various specialized tools have been developed since then [138], such as *PDP++* [123, 140], the *Neural Simulation Language* [185], *emergent* [139], the simulation platforms *DANA* [154], *TheVirtualBrain* [161], *Topographica* [8] and the *Neural Field Simulator* [133]. Similarly, efficient simulators for population-density approaches (*MIIND*: [40], *DiPDE*: [28]) as well as spiking neural networks (see [19] for a review) have evolved. The foci of the latter range from detailed neuron morphology (*NEURON*: [29], *GENESIS*: [15]) to an abstraction of neurons without spatial extent (*NEST*: see Subsec. 2.3.1 and Chapter 5, *BRIAN*: [68]). Such open-source software, combined with interfaces and simulator-independent languages [38, 47, 48], supports maintainability, reproducibility, and exchangeability of models and code, as well as community driven development. However, these tools are restricted to either rate-based or spike-based models only.

Currently, bottom-up and top-down strategies are still mostly disjointed. A major challenge in neuroscience is to form a bridge between the spike- and rate-based models [1], and, more generally, between the fields of computational neuroscience and cognitive science. From a practical point of view, a common simulation framework would allow the exchange and combination of concepts and code between the two descriptions and trigger interaction between the corresponding communities. This is in particular important since recent advances in simulation [49, 83, 91, 92, 108, 112] and computing technology [98, 126] enable full-density bottom-up models of complete circuits [119, 149]. In particular, it has become feasible to build spiking models [162] that describe the same macroscopic system as rate-based descriptions [31].

The relation between the different model classes is one focus of theoretical neuroscience. Assuming homogeneity across neurons, population-density methods reformulate the spiking dynamics as a dynamical equation for the probability density that captures the time evolution of the population activity [63, 64, 104]. Under certain assumptions allowing the neglect of fluctuations in the input to neurons, a set of coupled differential equations for the population-averaged firing rate and membrane potential can be derived [127]. For asynchronous irregular

activity, input fluctuations can be taken into account in a diffusion approximation, which leads to Fokker-Planck mean-field theory that can be used to determine homogeneous stationary state activities of spiking networks [20, 172]. The Fokker-Planck ansatz is, however, not limited to the population level, but can yield a heterogeneous stationary state firing rate across individual neurons in the network [159]. The dynamics of rate fluctuations around the background activity can be obtained using linear response theory on the population level [21] or the level of individual neurons [71, 116, 142, 164, 178], yielding effective rate models on the population or single-neuron level. An alternative to linear response theory is given by moment expansions for mode decompositions of the Fokker-Planck operator [44, 120, 121].

An alternative derivation of rate-based dynamics aims at a closure of equations for synaptic currents of spiking networks in a coarse-graining limit by replacing spiking input with the instantaneous firing rate [16]. Using field-theoretical methods [23] that were originally developed for Markovian network dynamics [24, 25] allows a generalization of this approach to fluctuations in the input [17].

In any case, the cascade of simplifications from the original spiking network to the rate-based model involves a combination of approximations which are routinely benchmarked in comparative simulations of the two models. A unified code base that features both models would highly simplify these validations, rendering duplication of code obsolete.

In many cases, rate models represent populations of spiking neurons. Thus, a hybrid model, employing both types of models in a multi-scale modeling approach, would contain a relatively large number of spiking neurons compared to the number of rate units. Despite the large size of the spiking network, the dynamics still features finite-size fluctuations [66, 85, 121, 125, 168], and a downscaling of the network can generally not be performed without changing correlations [179]. It is thus crucial that a common simulation framework is able to handle real-sized spiking networks. In addition, the employed mean-field theories exploit the large number of neurons in biological networks. In fact, they are strictly valid only in the thermodynamic limit $N \to \infty$ [86], where N denotes the number of units. Therefore, in the above mentioned validation studies, the spiking networks are typically large. Thus, a common simulation framework should be optimized for spiking neurons rather than rate-based models.

This chapter provides the concepts and a NEST implementation for the embedding of continuous-time dynamics in a spiking network simulator. Although rate-based models require communication of continuous state variables, they are in general compatible with the delayed $d_{\min}$-communication scheme of current spiking neural network simulators as long as these interactions have a delay. However, many rate-based models consider instantaneous interactions between

units (see [16] and references therein), typically for analytical convenience in quasi-static situations where delays do not matter. A priori, these interactions require communication between units at each time step and impair the performance and scalability of the simulators, especially on supercomputers where communication is particularly expensive because it is associated with a considerable latency. Therefore we introduce an (optional) additional iterative approach based on waveform-relaxation techniques that allows us to use the original $d_{\min}$-communication scheme also when instantaneous interactions are present. The scheme builds on the waveform-relaxation technique for SDEs from Chapter 4 and uses the framework already employed for gap-junction interactions in Chapter 5.

This chapter begins with a description of the class of rate models considered in this work in Sec. 6.1. Subsequently in Sec. 6.2, we describe the concepts for embedding rate-based network models into a simulation code for spiking networks. This includes the development of an extendable implementation framework for rate models in terms of templates and details on the usage of the waveform-relaxation method from Chapter 4. After that, Sec. 6.3 gives a brief summary of the user interface for rate models in NEST. The following numerical results section (Sec. 6.4) is structured as follows: first, different numerical schemes for SDEs are evaluated in order to find the most suitable method in the context of spiking neural network simulators. Then, the scalability of the NEST implementation is investigated with special focus on the comparison between the standard implementation and the use of the waveform-relaxation approach. Finally, the applicability of the framework to a broad class of network models is illustrated on the examples of a linear network model [71], a nonlinear network model [67, 174], a neural field model [155], and a mean-field description [188] of the stationary activity in a model of the cortical microcircuit [149, 166]. The chapter concludes with the discussion in Sec. 6.5. The technology described in this chapter was released with NEST 2.14.0 [145] and is available as open source under GNU General Public License (Version 2 or later). Some of the figures and parts of the text in this chapter have already been used in the original research article on the topic [72].

6.1 Rate models

We restrict our investigation to a class of rate models, which covers a large variety of rate models used in neuroscience today. We consider networks of N rate-based units where each unit receives recurrent input from the network. The system fulfills the Itô-SDEs

$$\begin{aligned}\tau_i dX_i(t) = &\left[-X_i(t) + \mu_i + \phi\left(\sum_{j=1}^{N} w_{ij}\psi\left(X_j(t-d_{ij})\right)\right)\right] dt \\ &+ \sqrt{\tau_i}\sigma_i\, dW_i(t) \qquad i = 1,\ldots,N\end{aligned} \tag{6.1}$$

with possibly nonlinear input-functions $\phi(x)$ and $\psi(x)$, connection weights w_{ij}, mean input μ_i, and optional delays $d_{ij} \geq 0$. The corresponding Fokker-Planck equation shows that the parameter $\sigma_i \geq 0$ controls the variance of $X_i(t)$ and the time constant $\tau_i > 0$ its temporal evolution. For readability, from here on we omit unit indices for σ, τ, μ, and d. The considered class of rate models only contains additive noise. Therefore we might as well interpret (6.1) as a system of Stratonovich-SDEs (see Sec. 2.2).

For illustrative purposes we explicitly state the different solution schemes presented in Subsec. 2.2.1 for the network dynamics (6.1) with $d = 0$. The Euler-Maruyama update step reads

$$X_{k+1,i} = X_{k,i} + \left[-X_{k,i} + \mu + \phi\left(\sum_{j=1}^{N} w_{ij}\psi\left(X_{k,j}\right)\right)\right]\frac{1}{\tau}\Delta t + \frac{1}{\sqrt{\tau}}\sigma\Delta W_{k,i}\,. \tag{6.2}$$

The implicit Euler update formula evaluates X (within the square brackets) at $k+1$ instead of k. This turns (6.2) into a system of nonlinear algebraic equations, which in fixed-point form is given as

$$X_{k+1,i} = \Phi_i\left(X_{k+1}\right) \tag{6.3}$$

with

$$\Phi_i\left(X_{k+1}\right) = \frac{X_{k,i} + \left[\mu + \phi\left(\sum_{j=1}^{N} w_{ij}\psi\left(X_{k+1,j}\right)\right)\right]\frac{1}{\tau}\Delta t + \frac{1}{\sqrt{\tau}}\sigma\Delta W_{k,i}}{1 + \Delta t/\tau}\,. \tag{6.4}$$

We can perform the fixed-point iteration

$$X_{k+1,i}^{(m+1)} = \Phi_i\left(X_{k+1}^{(m)}\right) \tag{6.5}$$

with initial value $X_{k+1}^{(0)} = X_k$ to obtain $X_{k+1,i}$, provided that (6.5) converges.

For nonlinear $\phi(x)$ or $\psi(x)$ the exponential Euler update step is

$$X_{k+1,i} = e^{-\Delta t/\tau} X_{k,i} + \left(1 - e^{-\Delta t/\tau}\right) \left[\mu + \phi\left(\sum_{j=1}^{N} w_{ij} \psi\left(X_{k,j}\right)\right)\right] + \sqrt{\frac{1}{2}(1 - e^{-2\Delta t/\tau})}\, \sigma \eta_{k,i} \tag{6.6}$$

with $\eta_{k,i} \sim \mathcal{N}(0,1)$. As in this case each rate unit (6.1) only contains a self-dependent linear part the exponential Euler scheme (2.20) does not rely on a matrix exponential, but decomposes into N equations with scalar exponential functions ($A = -I$ is a diagonal matrix). Note that with a linear choice, $\phi(x) = \psi(x) = x$, the system of SDEs can be written in matrix notation as

$$\tau dX(t) = \left[A \cdot X(t) + \mu\right] dt + \sqrt{\tau}\sigma I \, dW(t) \tag{6.7}$$

with $A = -I + W$ and $W = (w_{ij})_{N \times N}$. In this case the stochastic exponential Euler scheme (2.20) contains a matrix exponential and a matrix square root.

These numerical schemes can analogously be used for stochastic delay differential equations (SDDEs) ($d > 0$), if the delay d is a multiple of the step size Δt (see Subsec. 2.2.2). For the calculation of the approximation $X_{k+1,i}$ in time step $k+1$, the recurrent input is evaluated from $\frac{d}{\Delta t}$ steps earlier, i.e. from $X_{k-\frac{d}{\Delta t},j}$ for the explicit methods.

Characteristic features of the rate models considered in this chapter are the leaky dynamics, i.e the linear term $-X_i(t)$ in (6.1) and the additive coupling and noise. However, the framework to be presented later does not exclude the possibility to use more general rate models. It can easily be extended to rate models with

i) nonlinear dynamics

$$\tau dX_i(t) = \left[a(X_i(t)) + \phi\left(\sum_{j=1}^{N} w_{ij} \psi\left(X_j(t - d_{ij})\right)\right)\right] dt + \sqrt{\tau}\sigma dW_i(t)$$

where a characterizes the intrinsic rate dynamics, as for example used by Stern et al. [175].

ii) multiplicative coupling between units as, for example, employed in [60] or the original works of Wilson and Cowan [186, 187]. In the most general form, this amounts to

$$\tau dX_i(t) = \left[-X_i(t) + H(X_i) \cdot \phi\left(\sum_{j=1}^{N} w_{ij} \psi\left(X_j(t - d_{ij})\right)\right)\right] dt + \sqrt{\tau}\sigma dW_i(t)\,.$$

iii) multiplicative noise. The linear rate model considered in the numerical results section (especially in Subsec. 6.4.3.1) describes the dynamics around a stationary state and due to the stationary baseline, the noise amplitude is constant. However, one might relax the stationarity assumption, which would render the noise amplitude proportional to the time dependent rate, i.e. a multiplicative noise amplitude.

As a byproduct in this chapter we also implement rate models with so-called *output noise*. Grytskyy et al. [71] show that there is a mapping between a network of leaky integrate-and-fire models and a network of linear rate models with output noise. Here the noise is added to the output rate of the afferent units

$$\tau \frac{dX_i(t)}{dt} = -X_i(t) + \mu + \phi\left(\sum_{j=1}^{N} w_{ij}\psi\left(X_j(t-d_{ij}) + \sqrt{\tau}\sigma\xi_j(t)\right)\right) \tag{6.8}$$

for $i = 1, \ldots, N$ and we cannot write the equations as SDEs of type (2.13), as the nonlinearities $\phi(x)$ and $\psi(x)$ are also applied to the white noise ξ_j. For these models, the numerical methods considered here cannot be employed. Instead our solver assumes the noise ξ_j to be constant over the update interval which leads for $d = 0$ to the update formula

$$X_{k+1,i} = e^{-\Delta t/\tau} X_{k,i} + \left(1 - e^{-\Delta t/\tau}\right)\left[\mu + \phi\left(\sum_{j=1}^{N} w_{ij}\psi\left(X_{k,j} + \sqrt{\frac{\tau}{\Delta t}}\sigma\eta_{k,j}\right)\right)\right]. \tag{6.9}$$

The term $X_{k,j} + \sqrt{\frac{\tau}{\Delta t}}\sigma\eta_{k,j}$ with $\eta_{k,j} \sim \mathcal{N}(0,1)$ is calculated beforehand in the sending unit j.

6.2 Framework

This section describes the embedding of rate-based models of type (6.1) in a simulation code for spiking neuronal networks. The software architecture for rate models is based on existing concepts: Morrison et al. [129] describe distributed buffers for the storage of delayed interactions and the technique to consistently generate random numbers in a distributed setting, and our Sec. 5.1 introduces so-called *secondary events* that allow the communication of continuous state variables, like membrane potentials or rates, between neurons or rate units respectively (first published in [73]). Events provide an abstraction layer on top of the MPI communication, which allows the implementation of neuron models without explicit reference to MPI calls. Unlike primary events, which are used to transmit the occurrence of spikes at discrete points in time, secondary events occur on a regular time grid. These concepts are designed to be compatible with the parallel and distributed operation of a simulation kernel for spiking neuronal networks, ensuring an efficient use of clusters and supercomputers [83]. This allows researchers to easily scale up network sizes to more realistic number of neurons. The highly parallelizable structure of modern simulation codes for spiking neuronal networks, however, also poses restrictions on the utilizable numerical methods.

6.2.1 Restrictions

Parallelization of spiking neuronal networks is achieved by distributing neurons over compute nodes. Since the dynamics of spiking neurons (in the absence of gap junctions) is decoupled for the duration of the minimal synaptic delay $d_{\min}$ of the connections in the network, the states of the neurons can be propagated independently for this time interval. Thus it is sufficient to specify solvers on the single-neuron level. The spike times, i.e. the mediators of interaction between neurons, are then communicated in steps of $d_{\min}$.

As a result of this structure, the global connectivity of the network is unknown to the single neuron. The neuron object sends and receives events handled by an object on the compute node harboring the neuron termed network manager. However, the network manager only knows the incoming connections of the neurons on the compute node.

This poses restrictions on the use of implicit schemes. It is impossible to employ the implicit Euler scheme (2.18) with Newton iteration, since this would require the simultaneous solution of a system of nonlinear algebraic equations with information distributed over all compute nodes. The use of the implicit Euler scheme with fixed-point iteration is, however, compatible with this structure. To this end, the scheme (2.18) needs to be formulated as a fixed-point iteration

on the single-unit level (6.5) and the updated rates need to be communicated to the connected units after every iteration until some convergence criterion is met. The convergence of the fixed-point iteration is, however, only guaranteed if the scheme Φ is contractive (see e.g. [101], their Sec. 4.2), which (as shown below in Subsec. 6.4.1) limits the size of the step size Δt. Subsec. 6.4.1 investigates if the implementation can gain stability or accuracy from using the implicit Euler method with fixed-point iteration and if the payoff is large enough to justify the additional effort of an iterative solution scheme.

The restricted knowledge of connectivity also limits the usage of the exponential Euler method. In the case of a linear rate model, we are unable to add the influence from all other rate units to the matrix A in (6.7), because most of these connections are unknown at the single-unit level. Therefore, we use the exponential Euler method with $A = -I$ resulting in the update formula (6.6). This also has the benefit of avoiding the need to numerically evaluate a general matrix exponential, as A is a diagonal matrix.

6.2.2 Implementation

This section describes the additional data structure required for the implementation of rate-based models. While the naming convention refers to our implementation in NEST, the employed algorithms and concepts are portable to other parallel spiking network simulators. As a result of the previous section and our analysis of the numerical schemes in the numerical results section below (see in particular Subsec. 6.4.1) we restrict the final implementation in NEST to the exponential Euler method, where we assume $A = -I$ and identify $\Delta t = h$, with h denoting the global computation step size [129]. We have to distinguish the cases of connections with delay ($d > 0$) and connections without delay ($d = 0$). The former case is similar to spiking interaction: assuming a connection from unit i to unit j, the rate of unit i needs to be available at unit j after $\frac{d}{h}$ additional time steps. This can be ensured if the delay of the connection is considered in the calculation of the minimal delay $d_{\min}$ that determines the communication interval. After communication, the rate values are stored in a ring buffer of unit j until they are due [128]. This way we obtain the solution to the rate-neuron dynamics with delay (6.1) in a stepwise manner, as described in Subsec. 2.2.2. In the case of an instantaneous connection, the rate of unit i at time t_0 needs to be known at time t_0 at the process which updates unit j from t_0 to $t_0 + h$. Therefore, communication in every step is required for instantaneous rate connections, i.e. one has to set $d_{\min} = h$.

Due to the conceptual differences between instantaneous and delayed interactions, we employ two different connection types (`rate_connection_delayed` and `rate_connection_instantaneous`) and associated secondary events

(`DelayedRateConnectionEvent` and `InstantaneousRateConnectionEvent`). This distinction simplifies the discrimination of connections on the single-unit level, while still allowing for simultaneous use of instantaneous and delayed connections in the same rate model.

The large diversity of rate models (6.1) imposes a challenge for code maintenance and efficiency: each combination of nonlinearities $\phi(x)$ and $\psi(x)$ constitutes its own model. All of these models can be implemented in exactly the same way, except for the evaluation of the nonlinearities. A template class (`rate_neuron_ipn`), providing a base implementation for rate models of category (6.1), avoids code duplication. Nevertheless, we restrict the NEST implementation to one nonlinearity per model. This keeps the collection of rate models small while still covering the majority of rate models.

The template rate-model class is instantiated with an object that represents the nonlinearity. Being instantiated at compile time, this template solution does not incur additional overhead at run time compared to a solution using polymorphy (inheritance). A boolean class member `linear_summation` determines if the nonlinearity should be interpreted as $\phi(x)$ (true, default value) or $\psi(x)$ (false). The respective other function is assumed to be the identity function. The boolean parameter is evaluated in every update step of each unit. Deciding upon the type of nonlinearity at compile time would improve efficiency. In the present architecture this would, however, result in twice as many template instances for a given set of gain functions. With the future capabilities of code generation in NEST [148] in mind, it might be beneficial to elevate the constant boolean member object to a constant template parameter, to allow compilers efficient preprocessing and at the same time profit from the code reliability achievable by modern C++ syntax. The present base implementation reduces the effort of creating a specific rate model of category (6.1) to the specification of an instance of the template class. Afterwards, an actual rate model can be instantiated in a single line of code.

gain model	$\phi(x)$ or $\psi(x)$
`lin_rate`	$g \cdot x$ with $g \in \mathbb{R}$
`tanh_rate`	$\tanh(g \cdot x)$ with $g \in \mathbb{R}$
`threshold_lin_rate`	$g \cdot (x - \theta) \cdot H(x - \theta)$ with $g, \theta \in \mathbb{R}$

Table 6.1: **Template-derived rate-based models.** Gain functions of the rate-based models available in the NEST implementation. The name of a particular rate model is formed by `<gain model>_ipn`. The ending `ipn` indicates input noise, as the noise directly enters the r.h.s. of (6.1). H denotes the Heaviside function.

Table 6.1 gives an overview of template-derived rate models of the implementation in NEST. These models serve as examples for customized rate models. Activity of rate units can be recorded using the `multimeter` and the recordable `rate`.

In addition to these template-derived models of category (6.1), our implementation also contains a rate model called `siegert_neuron`. This model, described by (6.26) in Subsec. 6.4.3.4, is used for mean-field analysis of complex networks and constitutes a special case with respect to the recurrent input from the network. First, it requires a numerically stable implementation of the Siegert formula (see Appendix A.1 in [72]). Secondly, (6.24) and (6.25) in Subsec. 6.4.3.4 demonstrate that for this model the input rates are weighted by separate factors. Thus for connections between instances of this model, two different weights need to be specified and the rate model must be able to handle this anomaly. Therefore the `siegert_neuron` does not derive from our base class `rate_neuron_ipn`, but constitutes an independent class. It comes with connection type `diffusion_connection` providing the weight parameters. Sec. 6.3 below motivates the parameter names and shows the usage of the model in NEST.

6.2.3 Reduction of communication using waveform-relaxation techniques

Instantaneous connections between rate-based models require communication after every time step, thus in intervals of the global computation step size h. This requires setting $d_{\min} = h$ and impairs the performance and scalability, especially on supercomputers where communication is particularly expensive because it is associated with a considerable latency. Therefore, for simulations with instantaneous connections, we additionally study an iterative approach based on waveform-relaxation techniques that, upon convergence, produces the same results as the standard approach, but allows us to use communication on a coarser time grid.

In a simulator for spiking neuronal networks the minimal delay $d_{\min}$ in the network defines the communication interval. By employing the waveform-relaxation method with $\mathcal{T} = d_{\min}$, we retain this communication interval for simulations with instantaneous rate connections. To control the iteration interval $\mathcal{T}$ of the waveform-relaxation method, instantaneous connections contribute to the calculation of the minimal delay with an arbitrary user specified value given by the parameter `wfr_comm_interval` (see Table 6.2). Consequently, the actual communication interval for waveform relaxation then is $\mathcal{T} = \min(d_{\min}, \texttt{wfr\_comm\_interval})$.

The update step of a single rate unit in the m-th iteration of the waveform-relaxation method reads

$$X_{k+1,i}^{(m)} = e^{-\Delta t/\tau} X_{k,i}^{(m)} + \left(1 - e^{-\Delta t/\tau}\right) \left[\mu + \phi\left(\sum_{j=1}^{N} w_{ij} \psi\left(X_{k,j}^{(m-1)}\right)\right)\right] + \sqrt{\frac{1}{2}(1 - e^{-2\Delta t/\tau})}\, \sigma \eta_{k,i}\,. \tag{6.10}$$

Theorem 4.4 guarantees the full convergence of the scheme against the solution obtained with the standard approach after $\mathcal{T}/h$ iterations, regardless of the chosen step size h.

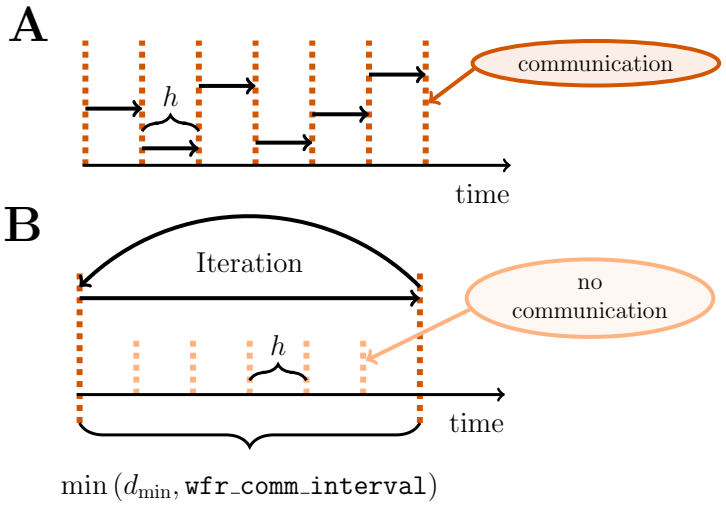

Figure 6.1: **Different communication strategies for distributed simulations.** Distance between neighboring dashed orange lines indicates computation time step of size h. Distance between neighboring dashed red lines symbolizes one communication interval where rates (and other events like spike events) are communicated at the end of the interval. **(A)** Standard solution for rate-based models: rates are communicated in every time step. **(B)** Iterative approach using waveform relaxation: rates are communicated only after $\frac{\mathcal{T}}{h}$ steps and the entire interval is solved repeatedly.

Fig. 6.1B illustrates the concept of the waveform-relaxation scheme in contrast to the standard procedure in panel A. The iterative approach requires the repeated solution of all time steps in the communication interval and converges to the solution obtained with the standard approach (Fig. 6.1A). The iteration terminates when a user chosen convergence tolerance `wfr_tol` (see Table 6.2) is met. If the method needs less than the maximal $\mathcal{T}/h$ iterations to reach the desired accuracy, the approach reduces the overall number of communications required to obtain the solution. In conclusion, the avoidance of communication in every step comes with the price of additional computational load.

The coupling of neurons via gap junctions from Chapter 5 is also instantaneous and continuous in time, and thus constitutes a very similar problem to the rate dynamics. In order to combine gap junctions with spiking dynamics, we have already used a waveform-relaxation method and devised a suitable framework, which is described in Sec. 5.1. This framework can also be employed for the simulation of rate-based models with instantaneous connections. The dynamics of a neuron model supporting gap junctions is solved with an adaptive step-size ODE-solver, routinely carrying out several steps of the employed numerical method within one

parameter name	type	default	description
`use_wfr`	`bool`	`true`	Boolean parameter to enable (`true`) or disable (`false`) the use of the waveform-relaxation technique. If disabled and any rate-based units (or neurons supporting gap junctions) are present, communication in every step is automatically activated ($d_{\min} = h$).
`wfr_comm_interval`	`double`	1.0 ms	Instantaneous rate connections (and gap junctions) contribute to the calculation of the minimal network delay with $\min(d_{\min}, \texttt{wfr\_comm\_interval})$. This way the length of the iteration interval of the waveform relaxation can be regulated.
`wfr_tol`	`double`	10^{-4}	Convergence criterion for waveform relaxation. The iteration is stopped if the rates of all units change less than `wfr_tol` from one iteration to the next.
`wfr_max_iterations`	`int`	15	Maximum number of iterations performed in one application of the waveform relaxation. If the maximum number of iterations has been carried out without reaching the accuracy goal, the algorithm advances system time and NEST issues a warning. Additional speed-up in the simulation of rate-based units can only be achieved by `wfr_max_iterations` $< \mathcal{T}/h$.
`wfr_interpolation_order`	`int`	3	This parameter is exclusively used for gap junctions (see Subsec. 5.1.1) and has no influence on the simulation of rate-based models.

Table 6.2: **Parameters of the waveform-relaxation algorithm.** The different parameters of the waveform-relaxation algorithm together with their C++ data-type, default value, and a brief description.

global computation time step h. The communication of a cubic interpolation of the membrane potential provides the solver with additional information, resulting in a more accurate solution than the one obtained from the standard approach. For rate-based models this additional benefit cannot be gained: the combination of an iterative method with an adaptive step-size solver is not applicable to SDEs, where the noise in each time step constitutes a random number. We thus use a fixed step size $\Delta t = h$ and need to ensure that the random noise applied to the units remains the same in every iteration. In Subsec. 6.4.2 we investigate the performance of the waveform-relaxation (Fig. 6.1B) and the standard approach (Fig. 6.1A) with a focus on large network simulations on supercomputers. In our implementation in NEST, waveform relaxation can be enabled or disabled by a

parameter `use_wfr`. Note that in the traditional communication scheme for spiking neuronal networks [129], the first communication occurs earliest at the end of the first update step. Therefore, in the absence of waveform relaxation, the initial input to units from the network is omitted.

Table 6.2 summarizes the parameters of our implementation of the waveform-relaxation technique. A subset (`wfr_interpolation_order`, `wfr_max_iterations`, `wfr_tol`) was previously introduced with the gap-junction framework described in Chapter 5, but we renamed them during the development of the rate-model framework to arrive at more descriptive names. The remaining parameters (`use_wfr`, `wfr_comm_interval`) result from the generalization to rate-based models.

6.3 User interface

We here give a brief description of how to use the implementation of the continuous-time dynamics in the simulation code NEST with the syntax of the PyNEST interface [51]. Complete simulation scripts are available in the corresponding NEST release 2.14.0 [145]. The description here focuses on rate model specific aspects.

Script 6.1: **Simulation of an excitatory-inhibitory network of linear rate units.** Here and in the following scripts we use the syntax of the PyNEST interface [51] of the NEST simulation software as of version 2.14.0 [145]. The script does not contain the definitions of the parameters (`h,NE,NI,mu,sigma,tau,T_start,w,d,g,KE,KI,T`).

```
import nest

# Disable usage of waveform-relaxation method
nest.SetKernelStatus({'resolution': h, 'use_wfr': False})

# Create rate units and recording device
n_e = nest.Create('lin_rate_ipn', NE,
                  params = {'linear_summation': True,
                  'mean': mu, 'std': sigma, 'tau': tau})
n_i = nest.Create('lin_rate_ipn', NI,
                  params = {'linear_summation': True,
                  'mean': mu, 'std': sigma, 'tau': tau})
mm = nest.Create('multimeter', params = {'record_from':
                 ['rate'], 'interval': h, 'start': T_start})

# Specify synapse and connection dictionaries
syn_e = {'weight': w, 'delay': d,
         'model': 'rate_connection_delayed'}
syn_i = {'weight': -g * w, 'delay': d,
         'model': 'rate_connection_delayed'}
conn_e = {'rule': 'fixed_outdegree', 'outdegree': KE}
conn_i = {'rule': 'fixed_outdegree', 'outdegree': KI}

# Connect rate units
nest.Connect(n_e, n_e, conn_e, syn_e)
nest.Connect(n_i, n_i, conn_i, syn_i)
nest.Connect(n_e, n_i, conn_i, syn_e)
nest.Connect(n_i, n_e, conn_e, syn_i)

# Connect recording device to rate units
nest.Connect(mm, n_e + n_i)

# Start simulation
nest.Simulate(T)
```

Script 6.1 shows the creation of an excitatory-inhibitory network of linear rate units. Researchers already familiar with the PyNEST interface will notice that there is no fundamental difference to scripts for the simulation of spiking neural networks. Line 4 illustrates how to disable the usage of the waveform-relaxation method. This is advisable for simulations on local workstations or clusters, where the waveform-relaxation method typically does not improve performance (see Subsec. 6.4.2). The usage of the method is enabled by default, as it is also employed for simulations with gap junctions, where it improves performance and even accuracy of the simulation, regardless of network size and parallelization (see Sec. 5.4). Instances of the linear rate-based model are created by calling `nest.Create` in the usual way with model type `lin_rate_ipn`. The parameter `linear_summation` characterizes the type of nonlinearity (ϕ or ψ, see Subsec. 6.2.2) of the rate model. In this particular example the explicit specification of the parameter is added for illustrative purposes only, as i) the employed model is a linear rate model, where regardless of the choice of the parameter, $\phi(x) = \psi(x) = x$ holds and ii) the default value is `True` anyway. Lines 13-14 and 31 demonstrate how to record the rate activity with the `multimeter`. The `record_from` parameter needs to be set to `rate` to pick up the corresponding state variable. As this particular network model includes delayed rate connections the synapse model `rate_connection_delayed` is chosen (lines 17-20). In order to create instantaneous rate connections instead, one changes the synapse model to `rate_connection_instantaneous` and removes the parameter `delay` from the synapse dictionary. For the simultaneous use of delayed and instantaneous connections one duplicates lines 17-28 and adapts the synapse and connection dictionaries of the copy according to the needs of the additional instantaneous connections.

Script 6.2: **Connecting units of type `siegert_neuron`.** The script shows how connections between units of type (6.26) are created in PyNEST. Again the code snippet does not contain the definitions of the parameters (`tau_m`,`K`,`w`) for brevity.

```
1 import nest
2
3 # Create one siegert_neuron for the excitatory population
4 s_ex = nest.Create('siegert_neuron', 1)
5 # Create one siegert_neuron for the inhibitory population
6 s_in = nest.Create('siegert_neuron', 1)
7
8 [...]
9
10 # Create connections originating from the excitatory unit
11 syn_e = {'drift_factor': tau_m * K * w,
12          'diffusion_factor': tau_m * K * w * w,
13          'model': 'diffusion_connection'}
14 nest.Connect(s_ex, s_ex + s_in, 'all_to_all', syn_e)
```

Script 6.2 shows a code snippet from a simulation script employing model (6.26) used for mean-field analysis of complex networks in Subsec. 6.4.3.4. Here, single rate units of type `siegert_neuron` represent an entire population of spiking neurons (lines 3-6). The units are coupled by connections of type `diffusion_connection`. This connection type is identical to type `rate_connection_instantaneous` for instantaneous rate connections except for the two parameters `drift_factor` and `diffusion_factor` substituting the parameter `weight` (lines 11-13). These two parameters reflect the prefactors in front of the rate variable in (6.24) and (6.25). In general the prefactors differ from these well known forms, as for example, in the case of distributed connection weights (see [84], their eq. 33). Therefore, we prefer a generic parameterization over more specific alternatives like the pair `weight` and `convergence`.

6.4 Numerical results

In the following section, we assess the accuracy and stability of different numerical solution schemes and benchmark the performance of the NEST implementation on large-scale machines, with special focus on scalability and the comparison between the standard solution and the iterative approach using waveform relaxation for simulations with instantaneous connections. The iterative approach is only discussed with respect to efficiency, as the iteration always converges to the results of the standard approach within only a couple of iterations (for details see Subsec. 6.2.3). The remainder of the section illustrates the application of the simulation framework to a selection of prominent problems in the neuroscientific literature.

6.4.1 Stability and accuracy of integration methods

In this section, we investigate numerical methods for the solution of SDEs that can be employed to solve the dynamics of rate-based units (see Sec. 6.1). We analyze the accuracy and stability of the different numerical methods to choose the best-suited method for application in a distributed simulation scheme of a spiking network simulation code. The analysis only covers methods compatible with a spiking neural network simulator, namely i) the Euler-Maruyama method, ii) the implicit Euler method solved with a parallelizable fixed-point iteration, and iii) the exponential Euler method where the linear part is restricted to be $-I$, from now on called *scalar exponential Euler*. The distributed representation of the global connectivity of the network rules out both the regular exponential Euler method with a nondiagonal matrix A and the implicit Euler method solved with Newton iteration (see Subsec. 6.2.1 for details).

We consider a network of N linear rate units with $\mu = 0$

$$\tau dX(t) = A \cdot X(t)\, dt + \sqrt{\tau}\sigma I\, dW(t)\,. \tag{6.11}$$

For this system of SDEs the regular exponential Euler scheme (2.20)

$$X_{k+1} = e^{A\Delta t/\tau} X_k + \sqrt{A^{-1}\left(\frac{e^{2A\Delta t/\tau} - I}{2}\right)} \cdot \sqrt{\tau}\sigma I \cdot \eta_k$$

produces a very accurate approximation to the exact solution. We analyze two test cases, i.e. two different choices of A, to demonstrate different stability constraints. First, an all-to-all connected network with inhibitory connections of weight $w_{ij} = \frac{-1}{\sqrt{N}}$ and hence $A = -I + \frac{-1}{\sqrt{N}} \cdot \mathbb{1}$, with $\mathbb{1}$ denoting an $N \times N$ all-ones matrix [32].

Secondly, a sparse balanced excitatory-inhibitory network where the number of excitatory units is four times larger than the number of inhibitory units. In this network, each unit receives input from a fixed number of $0.8 \cdot p \cdot N$ excitatory and $0.2 \cdot p \cdot N$ inhibitory randomly chosen source units with connection probability p and connection weights $\frac{1}{\sqrt{N}}$ and $\frac{-4}{\sqrt{N}}$, respectively. We will refer to the test cases as the *inhibitory all-to-all* and the *sparse balanced e/i* test case.

First, we turn to the accuracy analysis. Although the approximation produced by the regular exponential Euler scheme (2.20) cannot be obtained with a distributed representation of A, we can compute it using methods for numerical matrix computations implemented in MATLAB or Python (both provide an implementation of the same state-of-the-art algorithms, see [4, 41]). This way, we obtain a good reference solution $\widehat{X}$ for the computation of the root mean square error

$$\text{RMSE} = \sqrt{\frac{1}{NT} \sum_{i=1}^{N} \sum_{k=1}^{n} (\widehat{X}_{k,i} - X_{k,i})^2} \tag{6.12}$$

of the different approximate methods on the time interval $[t_0, t_0 + T]$. To employ the root mean square error in the context of stochastic differential equations, we determine the reference solution for every tested step size and use the same random numbers for both the reference solution and the approximative schemes.

Fig. 6.2 shows the root mean square error of the different numerical schemes for the two test cases with $N = 400$ units. In both test cases, all investigated methods with decreasing step size converge towards the reference solution with convergence order 1, which is consistent with the established theory for SDEs with additive noise [103]. Fig. 6.2A shows the results for the inhibitory all-to-all test case. Here, all three methods require a step size $\Delta t \leq 0.1$ to deliver reliable results. The implicit Euler scheme solved with fixed-point iteration even requires a step size $\Delta t \leq 0.05$. Within the stable region $\Delta t \leq 0.1$ the scalar exponential Euler scheme yields more accurate results than the two other methods. Fig. 6.2B shows the results for the sparse balanced e/i test case. Here all three methods achieve almost identical accuracy for $\Delta t \leq 0.1$ and stability problems only occur for the Euler-Maruyama method for step sizes $\Delta t > 0.5$. For step sizes $\Delta t > 0.1$, the implicit Euler method is more accurate than the other two methods.

To understand the stability issues shown in Fig. 6.2, we now turn to stability analysis. We assume that A is diagonalizable, i.e. $A = T^{-1}DT$ with $T = (t_{ij})_{N \times N} \in \mathbb{C}^{N \times N}$ and $D = \text{diag}(\lambda_1, \ldots, \lambda_N)$, and transform the system of SDEs with $Z(t) = T\,X(t)$. It follows that

$$\tau dZ(t) = D \cdot Z(t)\,dt + \sqrt{\tau}\sigma T\,dW(t)$$

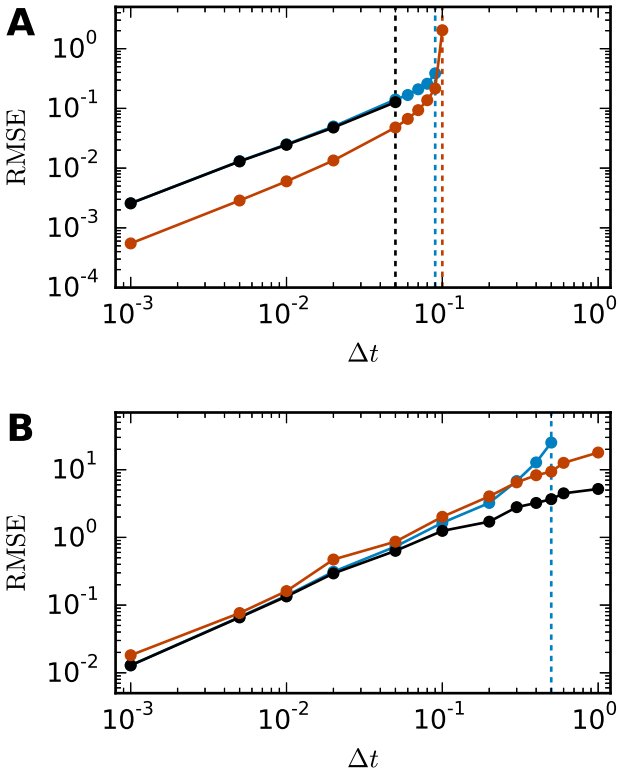

Figure 6.2: **Accuracy of numerical methods for two networks of linear rate units.** RMSE of the solution X obtained by the approximate solvers (blue curve: Euler-Maruyama method, black curve: implicit Euler method solved with fixed-point iteration, red curve: scalar exponential Euler method) with respect to the reference solution as a function of step size in double logarithmic representation. The respectively colored vertical lines mark the largest tested step size for which the corresponding methods deliver a solution with RMSE $\leq 10^{10}$. RMSE computed over 200.0 ms of biological time. **(A)** Inhibitory all-to-all test case. Network parameters: $N = 400$, $\mu = 0$, $\sigma = 10$ and $\tau = 1$ ms. **(B)** Sparse balanced e/i test case. Network parameters: $N = 400$, $p = 0.2$, $\mu = 0$, $\sigma = 10$ and $\tau = 0.5$ ms.

and $Z(t_0) = TX_0$. The transformed system consists of N equations of the form

$$\tau dZ_i(t) = \lambda_i \cdot Z_i(t)\, dt + \sum_{j=1}^{N} \sqrt{\tau}\sigma t_{ij}\, dW_j(t) \qquad i = 1, \ldots, n \tag{6.13}$$

that depend on the eigenvalues λ_i of A and are pairwise independent except for the common contributions of the Wiener processes $W_j(t)$. We will only consider networks with bounded activity, which requires eigenvalues $\lambda_i \in \mathbb{C}$ with negative real part, i.e. $\mathrm{Re}(\lambda_i) < 0$. For two different initial values $Z_{0,i}$ and $\tilde{Z}_{0,i}$ and assuming

the usage of the same random numbers, the solution of the i-th transformed equation then satisfies

$$|Z_i(t) - \tilde{Z}_i(t)| = e^{\lambda_i (t-t_0)/\tau} |Z_{0,i} - \tilde{Z}_{0,i}| < |Z_{0,i} - \tilde{Z}_{0,i}| \,. \tag{6.14}$$

It is a desirable stability criterion that a numerical method applied to (6.11) conserves this property. This requirement is closely related to the concept of A-stability for SDEs (see [103], their Chapter 9.8) as well as A- and B-stability for ODEs [76]. To derive stability conditions for the numerical schemes, we apply one update step (t_0 to $t_1 = t_0 + \Delta t$) of the methods to (6.13) and investigate under which circumstances the property $|Z_{1,i} - \tilde{Z}_{1,i}| < |Z_{0,i} - \tilde{Z}_{0,i}|$ is conserved. Here $Z_{1,i}$ denotes the approximation to the exact solution $Z_i(t_1)$ obtained by the numerical scheme. A straightforward calculation shows that the Euler-Maruyama method retains the property if $|1 + \lambda_i \cdot \Delta t/\tau| < 1$ holds. We conclude that the Euler-Maruyama scheme is stable for $\max_{\lambda_i} \zeta_{\mathrm{EM}}(\lambda_i) < 1$ with $\zeta_{\mathrm{EM}}(\lambda_i) = |1 + \lambda_i \cdot \Delta t/\tau|$.

The scalar exponential Euler method demands a splitting of $A = -I + W$ into $-I$ and W. Here, the stability condition is derived from the modified transformed system

$$\tau dZ_i(t) = (-1 + \tilde{\lambda}_i) \cdot Z_i(t)\, dt + \sum_{j=1}^{N} \sqrt{\tau} \sigma t_{ij}\, dW_j(t) \qquad i = 1, \ldots, n \tag{6.15}$$

where $\tilde{\lambda}_i$ are the eigenvalues of the matrix W. The scalar exponential Euler method conserves the property (6.14) if $|e^{-\Delta t/\tau} + \tilde{\lambda}_i (1 - e^{-\Delta t/\tau})| < 1$. We conclude that the scalar exponential Euler scheme is stable for $\max_{\lambda_i} \zeta_{\mathrm{EXP}}(\lambda_i) < 1$ with $\zeta_{\mathrm{EXP}}(\lambda_i) = |1 + \lambda_i \cdot (1 - e^{-\Delta t/\tau})|$.

The implicit Euler method solved with fixed-point iteration is stable, provided the fixed-point iteration converges. For the transformed system, the employed fixed-point iteration on the single-unit level (6.4) reads

$$\Phi_i(Z_{k+1,i}^{(m)}) = \frac{1}{1 + \Delta t/\tau} (Z_{k,i} + \tilde{\lambda}_i\, Z_{k+1,i}^{(m)}\, \Delta t/\tau + \sum_{j=1}^{N} \frac{1}{\sqrt{\tau}} \sigma t_{ij} \Delta W_{k,j}) \,.$$

It converges if the scheme Φ_i is contractive, i.e. if the inequality

$$|\Phi_i(Z_{k+1,i}^{(m)}) - \Phi_i(\tilde{Z}_{k+1,i}^{(m)})| \;\; \leq \;\; C \cdot |(Z_{k+1,i}^{(m)} - \tilde{Z}_{k+1,i}^{(m)})|$$

holds for all m and any two initial values $Z_{k+1,i}^{(0)}$ and $\tilde{Z}_{k+1,i}^{(0)}$ with a constant $C < 1$. It follows that the fixed-point iteration converges if $|\frac{\Delta t/\tau}{1 + \Delta t/\tau} \tilde{\lambda}_i| < 1$. Thus the

implicit Euler method solved with fixed-point iteration is stable if $\max_{\lambda_i} \zeta_{\mathrm{IE}}(\lambda_i) < 1$ with $\zeta_{\mathrm{IE}}(\lambda_i) = \frac{\Delta t/\tau}{1+\Delta t/\tau}|\lambda_i + 1|$. Hence for all investigated methods the stability depends on the eigenvalues of the matrix A, the time constant τ and the step size Δt. To conclude restrictions on the step size Δt, we therefore analyze the eigenvalues of A for our examples.

For the inhibitory all-to-all test case we determine the eigenvalues $\lambda_1 = -1 - \sqrt{N}$ and $\lambda_2 = \ldots = \lambda_N = -1$ of the matrix $A = -I + \frac{-1}{\sqrt{N}} \cdot \mathbb{1}$ analytically. It follows that the Euler-Maruyama scheme satisfies the stability criterion for $\Delta t \leq \frac{2\tau}{\sqrt{N}+1}$, the scalar exponential Euler method demands $\Delta t \leq -\tau \cdot \ln(\frac{\sqrt{N}-1}{\sqrt{N}+1})$ and the implicit Euler method with fixed-point iteration requires $\Delta t \leq \frac{\tau}{\sqrt{N}-1}$. For the example of 400 units with $\tau = 1$ in Fig. 6.2A, this yields step size restrictions of $\Delta t \leq \frac{2}{21}$ for the Euler-Maruyama method, $\Delta t \leq -\ln(\frac{19}{21}) \approx 0.1$ for the scalar exponential Euler method and $\Delta t \leq \frac{1}{19}$ for the implicit Euler method. This is consistent with the numerically obtained result (see vertical lines). For all methods, the stability criterion implies that the step size Δt needs to be reduced with increasing network size N or decreasing time constant τ.

This fully connected network, however, constitutes the worst case test for the class of rate-based models (6.1), as the absolute value of the negative eigenvalue quickly increases with the number of units N. Our second test case, the sparse balanced e/i network does not suffer from this problem, as it is a perfectly balanced network of excitatory and inhibitory units. In a scaling of the connection weights as $\frac{1}{\sqrt{N}}$, the spectral radius $\rho(A)$ of A and therefore the subsequent stability analysis is independent of N (see [150]) and only depends on the connection probability p. Here, the stability analysis is more complicated, as most of the eigenvalues of A are complex and we compute them numerically. Fig. 6.3A shows the eigenvalues of A for a network of 2000 units with $p = 0.2$. Fig. 6.3B demonstrates that for this test case, the scalar exponential Euler method and the implicit Euler method are stable regardless of the step size Δt. For the Euler-Maruyama the step size is restricted to $\Delta t < 1.03\tau$. This is again consistent with the results obtained in Fig. 6.2B, where $\tau = 0.5$ and therefore the stability criterion of the Euler-Maruyama method yields $\Delta t < 0.515$.

Random inhibition-dominated networks exhibit characteristics of both examples. First, the matrix A contains a real eigenvalue $\lambda_1 = -1 - \alpha\sqrt{N}$ which scales with the network size, however, it scales with a proportionality constant $0 < \alpha < 1$, which is reduced compared to the fully connected inhibitory network and determined by the sparseness and the imbalance between excitation and inhibition. Secondly, the matrix A contains eigenvalues which constitute a cloud in the complex plane that is determined by the randomness of the connectivity. For these random networks $\lambda_{\max} = \mathrm{argmax}_{\lambda_i} \zeta(\lambda_i)$ is a real eigenvalue.

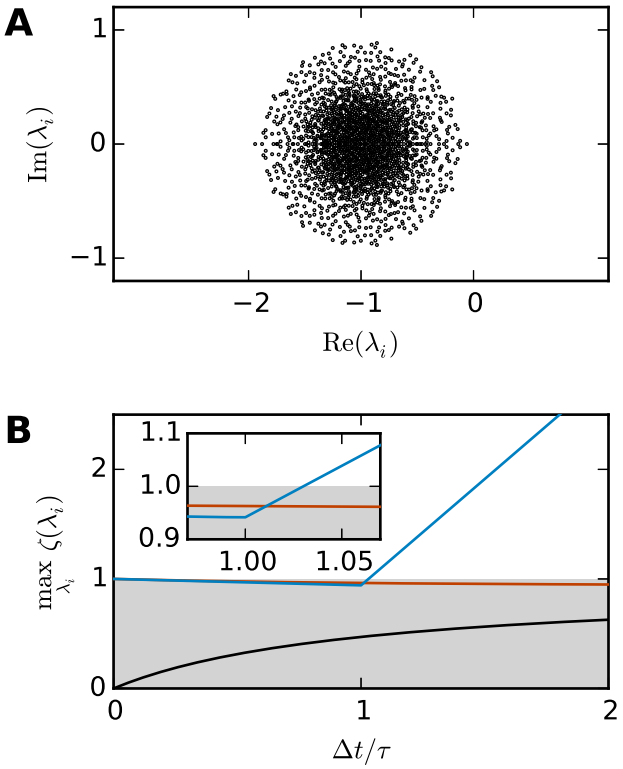

Figure 6.3: **Stability analysis for the sparse balanced e/i test-case. (A)** Black circles show the eigenvalues λ_i of the matrix A. Network parameters: $N = 2000$, $p = 0.2$. **(B)** The curves show the maximum of the stability function $\zeta(\lambda_i)$ over all eigenvalues λ_i for the investigated methods (ζ_{EM}: blue, ζ_{IE}: black, ζ_{EXP}: red) with respect to $\Delta t/\tau$. The gray area indicates the region where the stability criterion is met.

Fig. 6.4 shows the step size restrictions of the different numerical methods with respect to the absolute value of $\lambda_{\max}$. For $|\lambda_{\max}| < 2$ the scalar exponential Euler scheme and the implicit Euler scheme are stable regardless of the step size Δt. Starting at $|\lambda_{\max}| \geq 2.8$, the scalar exponential Euler scheme is more stable than the implicit Euler method solved with fixed-point iteration. With increasing $|\lambda_{\max}|$ the step size restrictions of the scalar exponential Euler method converges to the step size restriction of the Euler-Maruyama method.

Based on the results in this section we choose the scalar exponential Euler to solve rate-based model dynamics (6.1) in our NEST implementation. Fig. 6.4 demonstrates that it is the most stable algorithm compatible with the constraints of the distributed simulation scheme for spiking neural networks. Furthermore, the results in Fig. 6.2 indicate that it is the most accurate method in the case of an all-to-all connected network with inhibitory connections. For the sparse balanced excitatory-inhibitory network, the analysis exhibits an accuracy similar

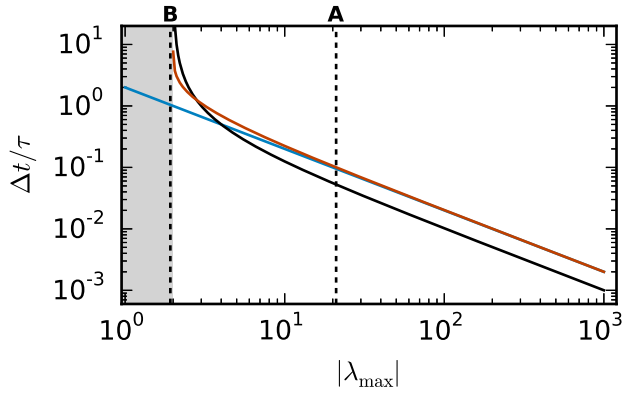

Figure 6.4: **Step size restrictions of the numerical methods.** Largest ratio $\Delta t/\tau$ for which the different numerical methods (blue curve: Euler-Maruyama method, black curve: implicit Euler method solved with fixed-point iteration, red curve: scalar exponential Euler method) are stable shown against the absolute value of $\lambda_{\max} = \text{argmax}_{\lambda_i} \zeta(\lambda_i) \in \mathbb{R}$. The gray area indicates the region where the scalar exponential Euler method and the implicit Euler method are stable without any restrictions on $\Delta t/\tau$. The dashed vertical lines correspond to the examples presented in the panels of Fig. 6.2.

to the implicit Euler method. However, the solution of the implicit Euler method with fixed-point iteration requires the application of an iterative scheme in each single time step, with communication between the units after every iteration. This algorithm is therefore more time consuming than the scalar exponential Euler scheme. Besides the choice of method, the analysis in this section indicates that numerical stability is an issue for all tested methods depending on step size Δt and time constant τ. Although the applications in Subsec. 6.4.3 show that many practical examples do not suffer from stability issues when a commonly used simulation step size is employed, the inevitable restrictions on the step size Δt should be taken into account in simulations of rate-model networks. For simulations of linear rate models, an appropriate step size can be determined by an analysis of the eigenvalues of A.

6.4.2 Performance of the NEST implementation

This section investigates the performance of the rate-model implementation in NEST. We are interested in i) the scalability of the rate-model framework and ii) the comparison between the standard implementation with communication in every computation time step and the iterative approach using waveform relaxation (see Subsec. 6.2.3 for details). We perform the simulations on the JUQUEEN BlueGene/Q supercomputer [98] at the Jülich Research Centre in Germany (see Subsec. 5.4.1.2 for details on JUQUEEN). As a test case we employ the sparse

balanced e/i test case of linear rate units ($\phi(x) = \psi(x) = x$) introduced in Subsec. 6.4.1, but with a fixed number of 2000 inputs per unit independent of the overall number of units to allow for an unbiased weak scaling. Thus the connection probability p is decreasing with increasing overall network size.

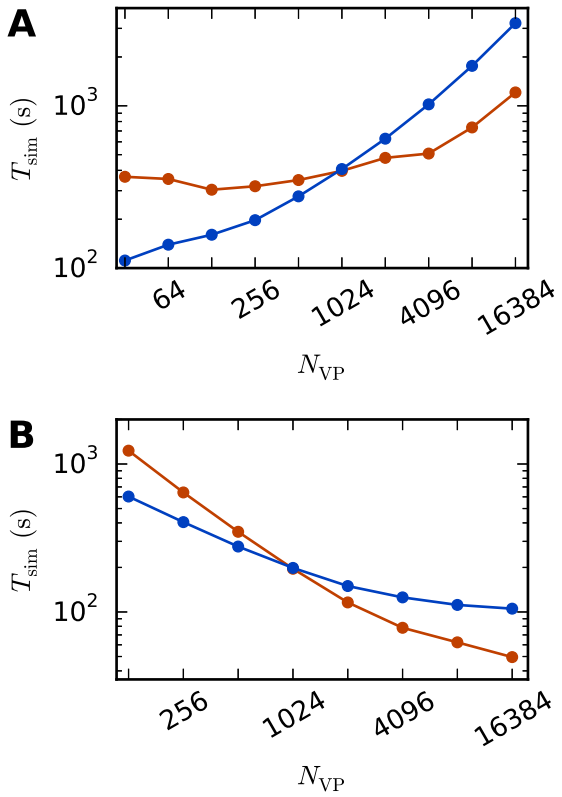

Figure 6.5: **Scaling behavior of an excitatory-inhibitory network.** Simulation time with waveform relaxation (red curves, `wfr_comm_interval`: 1.0 ms, `wfr_tol`: 10^{-4}) and without waveform relaxation (blue curves) as a function of the number of virtual processes in double logarithmic representation. The simulations span 100 ms of biological time at a computation step size of $h = 0.1$ ms. Sparse balanced e/i test case but with a fixed number of 2000 inputs per unit. Other parameters: $\mu = 0$, $\sigma = 1$ and $\tau = 10$ ms **(A)** Weak scaling with 100 units per virtual process. **(B)** Strong scaling with a total number of $N = 51,200$ units.

A weak scaling (Fig. 6.5A) shows that the scalability of the standard implementation is impaired by the massive amount of communication. While for perfect scaling the simulation time should be constant over the number of virtual processes, the actual simulation time is increased by $15 - 25\%$ each time the number of virtual processes is doubled for $N_{\mathrm{VP}} < 256$ and further up to 83% when we go from $8,192$ to $16,384$ virtual processes. For the waveform-relaxation method, the scaling behavior is close to constant up to $1,024$ virtual processes. When more

processes are employed, the simulation time increases. However, the waveform-relaxation method shows better scaling behavior, as the increase is weaker compared to the standard approach, due to the lower total number of communication steps. Due to the higher computational load of the iterative method (see Subsec. 6.2.3), the simulation time is larger compared to the straight forward approach for a small number of virtual processes, where communication is not that crucial. For $N_{\mathrm{VP}} \geq 1024$, the iterative approach is superior with a speed up factor close to three for $16,384$ virtual processes (1209 s vs. 3231 s).

The strong scaling scenario with a fixed total number of $N = 51,200$ units in Fig. 6.5B gives a similar result. The iterative approach is beneficial for more than $1,024$ virtual processes and the scaling behavior of the iterative method outperforms that of the standard computation. Starting at $4,096$ virtual processes, the savings in computation time decrease, which is explained by the very low workload of each single compute node. Again, for a smaller number of virtual processes the amount of additional computations is too high to outperform the standard implementation.

Despite the overall good scaling behavior, the performance in terms of absolute compute time is inferior to a simulator specifically designed for rate-based models alone (not shown). In such simulators it increases performance if one collects the states of all units in one vector. If furthermore the connectivity is available in form of a matrix and the delays are zero or homogeneous, the network can be efficiently updated with a single matrix-vector multiplication. Thus the increased functionality and flexibility of having rate- and spiking models unified in one simulator comes with the price of a loss of performance for the rate-based models. However, as noted in the introduction of this chapter, the number of units in rate-based network models is usually small and therefore performance is not as critical as for spiking network models.

Nevertheless, we want to develop a better understanding of the conditions under which the use of the waveform-relaxation method is beneficial for large-scale simulations of rate-based networks. Obvious requirements are i) that the overall number of communications is reduced by the method and ii) that the communication of NEST on the computing system is expensive enough that the additional computational effort is justified. While Fig. 6.5 demonstrates that the latter is true for JUQUEEN and that the former is true for the considered test case, we are interested in whether the former requirement is also met for other linear rate-unit networks.

For the analysis we employ the error bound (4.19) from Theorem 4.8. For networks of linear rate units of type (6.11), we identify the constants in (4.19) as $a = 1/\tau$ and $C = \|W\|/\tau$, where $\|\cdot\|$ denotes a matrix norm that is compatible with the vector norm used for the measurement of the error $e_m = \max_{0 \leq k \leq \mathcal{T}/\Delta t} \|X_k^{(m)} -$

$X_k\|$. From standard numerical analysis (see e.g. [176], their Chapter 6.9) we know that $\|W\| \geq \rho(W)$ holds for every matrix norm and that there exists a vector norm $\|\cdot\|_\epsilon$ for each matrix W such that the induced matrix norm satisfies $\rho(W) \leq \|W\|_\epsilon \leq \rho(W) + \epsilon$. Choosing this ϵ-norm with ϵ very close to zero for the calculation of the error bound thus enables us to estimate C by the eigenvalues $\tilde{\lambda}_i = \lambda_i + 1$ of W. The downside of this choice is that we cannot compute the ϵ-norm for very small ϵ due to numerical instabilities, and thus cannot compute the initial error e_0. We can, however, observe the reduction of the initial error over the iterations, which is also a good indicator for the convergence speed.

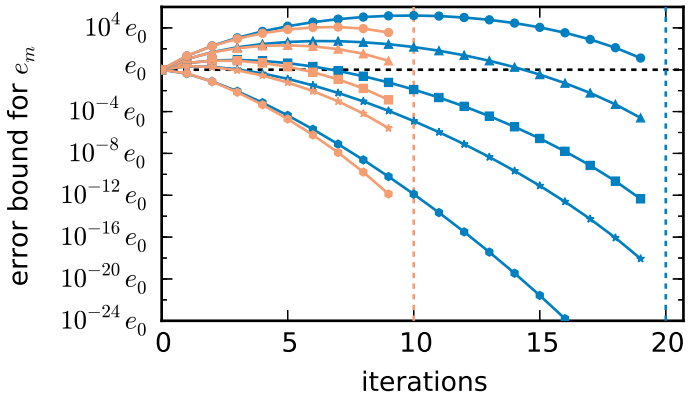

Figure 6.6: **Error bound for the error $\mathbf{e_m = \max_{0 \leq k \leq \mathcal{T}/\Delta t} \|X_k^{(m)} - X_k\|}$.** Evolution of the error bound for e_m (see Theorem 4.8, (4.19)) depending on the number of iterations. Orange curves display the bound for step size $\Delta t = 0.1$ and blue curves for $\Delta t = 0.05$ at an iteration interval length of $\mathcal{T} = 1.0\,\text{ms}$. The dashed vertical lines indicate the number of iterations where full convergence is reached for the corresponding step size. Different markers belong to different combinations of a and C in error bound (4.19) ($a = 1$ and $C = 20$ (circles), $a = 1$ and $C = 10$ (triangles), $a = 2$ and $C = 2$ (squares), $a = 2$ and $C = 1$ (stars) and $a = 2$ and $C = 0.2$ (hexagons)).

Fig. 6.6 shows the error bound for different combinations of a and C. The combinations $a = 1$ and $C = 10$, respectively $C = 20$ correspond to the inhibitory all-to-all test case with $\tau = 1\,\text{ms}$ and $N = 100$ and $N = 400$. In both setups, the error bound indicates that the error e_m increases over the first couple of iterations for both tested step sizes. A reduction of the initial error e_0 earlier than after $\mathcal{T}/h$ iterations, where full convergence is reached, can only be guaranteed for the case of $N = 100$ simulated with step size $h = \Delta t = 0.05\,\text{ms}$. As C quickly increases with the network size the behavior is even worse for larger networks. Thus for the inhibitory all-to-all test case, the usage of the waveform-relaxation method does not pay off.

The combination $a = 2$ and $C = 2$ corresponds to the sparse balanced e/i test case

with $p = 0.2$ and $\tau = 0.5\,\text{ms}$. As indicated by Fig. 6.3 the connection probability $p = 0.2$ results in a spectral radius $\rho(W) \approx 0.95$ and $\|W\|_\epsilon$ can therefore be safely estimated by one, regardless of the network size N. Here a first reduction of the initial error can be guaranteed after six of maximally ten iterations for $\Delta t = h = 0.1\,\text{ms}$, and after eight of maximally 20 iterations for $\Delta t = h = 0.05\,\text{ms}$. The actual number of iterations in a network simulation in NEST is determined by the magnitude of the initial error e_0 and the desired accuracy `wfr_tol`. A reduction of the initial error by a factor of 10^{-5} as achieved after 15 iterations with step size 0.05 ms might, however, be sufficient and indicates that the number of communications can be reduced with the waveform-relaxation method for this test case.

The remaining combinations with $a = 2$ and smaller values for C correspond to the sparse balanced e/i test case with a fixed number of inputs per unit investigated in Fig. 6.5. In the limit of large networks, this is the most realistic test case, as the number of incoming connections of a single unit is usually limited by some biologically justified threshold. The result of a fixed number of inputs at an increasing network size is a more sparse matrix W and a decreasing connection probability p. Numerically, we obtain the spectral radius of sufficiently large matrices W as $\rho(W) \approx 0.47$ for $p = 0.05$ and as $\rho(W) \approx 0.075$ for $p = 0.001$, which motivate the remaining choices of C in Fig. 6.6. In the latter case, the initial error is already reduced by a factor of 10^{-5} or 10^{-12} after half of the maximal number of iterations. The error bound thus gives another reason besides expensive communication as to why the waveform-relaxation method is particular beneficial with a large number of virtual processes N_{VP} in the weak scaling shown in Fig. 6.5A: the waveform-relaxation method is expected to need fewer iterations if the network size is increased, and thus the connection probability p is reduced. However, as the strong scaling with a fixed connection probability in Fig. 6.5B shows a very similar behavior, the expensive communication still is the main criterion here.

In conclusion the convergence speed of the waveform-relaxation method depends on the time constant τ and the eigenvalues of the matrix W. Fast convergence can be obtained for networks with large time constant τ and a matrix W with a small spectral radius $\rho(W)$. Following the stability analysis in Subsec. 6.4.1, those networks are exactly the kind of networks which can be simulated in NEST without or with only moderate step size restrictions due to stability issues.

6.4.3 Applications

This section demonstrates the use of the framework on different neuroscientific applications. First, we discuss a random inhibition-dominated network of linear rate units, then include nonlinear rate dynamics in a random network and spatially structured connectivity in a functional neural-field model. In each case, simulation results are compared to analytical predictions. Furthermore, we simulate a mean-field model of a spiking model of a cortical microcircuit and discuss possible generalizations.

6.4.3.1 Linear model

In the asynchronous irregular regime which resembles cortical activity, the dominant contribution to correlations in networks of nonlinear units is given by effective interactions between linear response modes [37, 71, 144, 178]. Networks of such noisy linear rate models have been investigated to explain features such as oscillations [13] or the smallness of average correlations [85, 177]. We here consider a prototypical network model of excitatory and inhibitory units following the linear dynamics given by (6.1) with $\phi(x) = \psi(x) = x$, $\mu = 0$, and noise amplitude σ,

$$\tau dX_i(t) = \left(-X_i + \sum_{j=1}^{N} w_{ij} X_j(t) \right) dt + \sqrt{\tau}\sigma dW_i(t)\,. \tag{6.16}$$

Due to the linearity of the model, the cross-covariance between units i and j can be calculated analytically and is given by [37, 61, 66, 152]

$$c(t) = \sum_{i,j} \frac{v_i^T \sigma^2 v_j}{\lambda_i + \lambda_j} u_i u_j^T \left(H(t)\frac{1}{\tau} e^{-\lambda_i \frac{t}{\tau}} + H(-t)\frac{1}{\tau} e^{\lambda_j \frac{t}{\tau}} \right), \tag{6.17}$$

where H denotes the Heaviside function. The λ_i indicate the eigenvalues of the matrix $I - W$ corresponding to the i-th left and right eigenvectors v_i and u_i respectively. Non-zero delays yield more complex analytical expressions for cross-correlations. In the population-averaged case, theoretical predictions are still analytically tractable ([71], their eq. 18). Fig. 6.7 shows the cross-covariance functions for pairs of instantaneously coupled units in a large network, as well as population-averaged covariance functions in a network of excitatory and inhibitory units with delayed interactions. In both cases, simulations are in good agreement with the theoretical predictions.

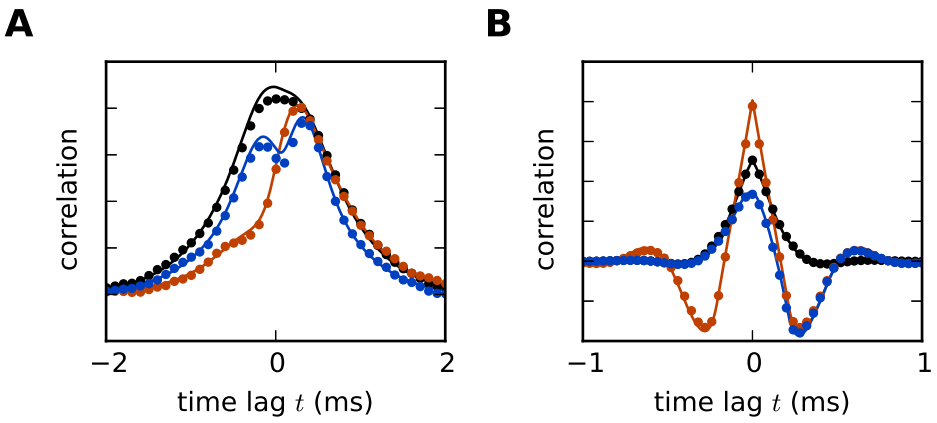

Figure 6.7: **Linear rate model of a random excitatory-inhibitory network. (A)** Cross-correlation functions of two pairs of excitatory units (black, red) and an excitatory-inhibitory unit pair (blue) in a network without delay. The variability across correlation functions arises from heterogeneity in network connections (difference between black and red curves) and from different combinations of cell types (e.g. difference between black and blue curves). **(B)** Population-averaged autocorrelation function for excitatory (black) and inhibitory units (red), and cross-correlation function between excitatory and inhibitory units (blue) in a network with delay $d = 2\,\text{ms}$. Symbols denote simulation results, curves show theoretical predictions. Parameters: $N_E = 80$ excitatory and $N_I = 20$ inhibitory units, random connections with fixed out-degree, connection probability $p = 0.1$, excitatory weight $w_E = 1/\sqrt{N_E + N_I}$, inhibitory weight $w_I = -6\,w_E$, $\tau = 1\,\text{ms}$, $\mu = 0$, $\sigma = 1$. Step size $h = 0.1\,\text{ms}$.

6.4.3.2 Nonlinear model

In the presence of nonlinearities qualitatively new features appear. One of the most prominent examples is the emergence of chaotic dynamics [174] in a network of non-linearly coupled rate units. The original model is deterministic and has been recently extended to stochastic dynamics [67]. The model definition follows from (6.1) with $\mu = 0$, $\phi(x) = \text{x}$, $\psi(x) = \tanh(x)$, i.e.

$$\tau dX_i(t) = \left(-X_i(t) + \sum_{j=1}^{N} w_{ij} \tanh\left(X_j(t)\right) \right) dt + \sqrt{\tau}\sigma\, dW_i(t)\,, \tag{6.18}$$

where $w_{ij} \approx \mathcal{N}(0, g^2/N)$ are Gaussian random couplings. In the thermodynamic limit $N \to \infty$, the population averaged autocorrelation function $c(t)$ can be determined within dynamic mean-field theory [67, 165, 174]. Comparing $c(t)$ obtained by simulation of a network (6.18) with the analytical result ([67], their eqs. 6 and 8) demonstrates excellent agreement (Fig. 6.8). The simulated network is

two orders of magnitude smaller than the cortical microcircuit, illustrating that in this context finite-size effects are already negligible at this scale.

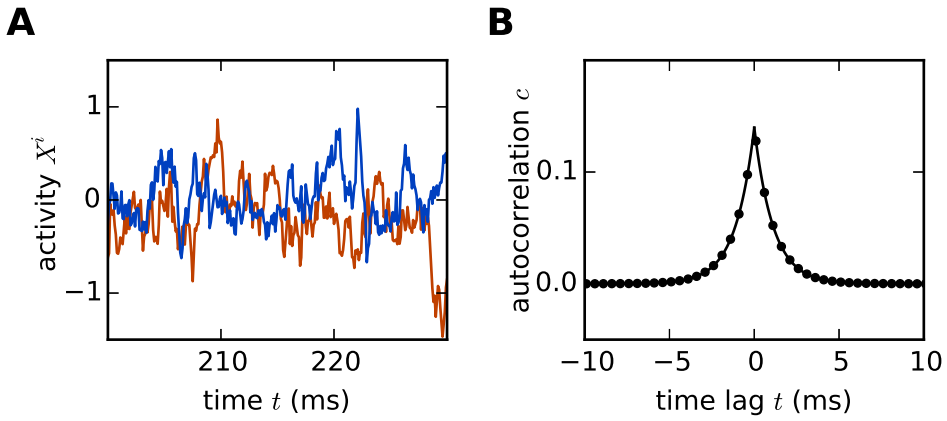

Figure 6.8: **Nonlinear network model.** Simulation of the network specified by (6.18) with $N = 1000$ units. **(A)** Noisy example trajectories of two units. **(B)** Autocorrelation function obtained by simulation averaged over all units (dots) and theory (solid curve). Other parameters: $\tau = 1\,\text{ms}$, $\sigma = 0.5$, $g = 0.5$. Step size $h = 0.1\,\text{ms}$.

6.4.3.3 Functional model

Complex dynamics arises not only from nonlinear single-unit dynamics but also from structured network connectivity [189]. One important nonrandom feature of brain connectivity is the spatial organization of connections [117, 183]. In spatially structured networks, delays play an essential role in shaping the collective dynamics [155, 182]. Patterns of activity in such networks are routinely investigated using neural-field models. In contrast to the models discussed above, field models require a discretization of space for numerical simulation. Such discretization can be done in the real space, leading effectively to a network of units at discrete positions in space, or alternatively, for particular symmetries in the couplings in k-space [156]. Here, we follow the more general approach of discretization in real space.

A prototypical model of a spatial network is given by Roxin et al. [155], where the authors consider the neural-field model

$$\tau dX(\varphi,t) = \left(-X(\varphi,t) + \phi\left[I_{\text{ext}} + \int_{-\pi}^{\pi} d\varphi'\, w(|\varphi-\varphi'|)X(\varphi',t-d)\right]\right) dt \quad (6.19)$$

with delayed (delay d) interactions, constant input I_{ext}, threshold-linear activation function $\phi = x \cdot H(x)$ and periodic Mexican-hat shaped connectivity

$$w(|\varphi-\varphi'|) = w_0 + w_1 \cos(\varphi-\varphi'). \quad (6.20)$$

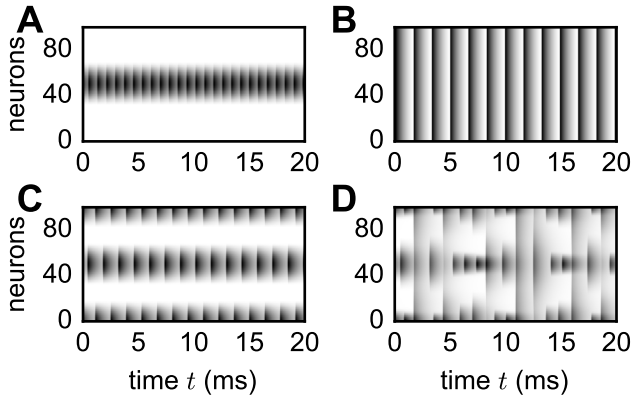

Figure 6.9: **Spatial patterns in functional neural-field model.** Vertical axis shows unit indices organized according to ascending angle $\varphi \in [-\pi, \pi)$. Activity $X_i(t) = X(\varphi_i, t)$ encoded by gray scale with white denoting no activity. Initial transients not shown. Patterns reproduce the analytically derived phase diagram in the original study by Roxin et al. [155]. Parameters: $N = 100$, $d = 0.1\,\text{ms}$, $\tau = 1\,\text{ms}$, $I_{\text{ext}} = 1$, $w_0 = -80$, $w_1 = 15$ (**A**), $w_1 = 5$ (**B**), $w_1 = -46$ (**C**), $w_1 = -86$ (**D**). Initial condition: $X_i(0) = X(\varphi_i, 0) = \pi^2 - \varphi_i^2$.

The spatial variable φ can also be interpreted as the preferred orientation of a set of units, thus rendering (6.19) a model in feature space [80]. Discretizing space into N segments yields the following set of coupled ODEs

$$\tau dX^i = \left(-X^i + \phi \left[I_{\text{ext}} + \sum_{j=1}^{N} w^{ij} X^j(t-d) \right] \right) dt \tag{6.21}$$

with connectivity $w^{ij} = \frac{2\pi}{N} w(|\varphi^i - \varphi^j|)$, $\varphi_i = -\pi + \frac{2\pi}{N} \cdot i$ for $i \in [1, N]$ and discretization factor $\frac{2\pi}{N}$ that scales the space constants w_0 and w_1 with the neuron density. The spatial connectivity together with a delay in the interaction introduces various spatial activity patterns depending on the shape of the Mexican-hat connectivity. To illustrate applicability of the simulation framework to neural-field models, we reproduce various patterns (Fig. 6.9) observed by Roxin et al. [155]. Although the discrete and continuous networks strictly coincide only in the thermodynamic limit $N \to \infty$, the numerically obtained patterns shown in Fig. 6.9 well agree with the analytically derived phase diagram of the continuous model [155] already for network sizes of only $N = 100$ units.

6.4.3.4 Mean-field analysis of complex networks

A network of spiking neurons constitutes a high dimensional and complex system. To investigate its stationary state, one can describe the activity in terms

of averages across neurons and time, leading to population averaged stationary firing rates [20]. Here, the spatial average collapses a large number of neurons into a single population, which is interpreted as a single rate unit. The ability to represent spiking as well as rate dynamics by the same simulation framework allows a straight-forward analysis of the spiking network by replacing the spiking neuron populations with single rate-based units.

In more formal terms, we now consider networks of neurons structured into N interconnected populations. A neuron in population α receives $K_{\alpha\beta}$ incoming connections from neurons in population β, each with synaptic efficacy $w_{\alpha\beta}$. Additionally, each neuron in population α is driven by $K_{\alpha,\mathrm{ext}}$ Poisson sources with rate X_{ext} and synaptic efficacy w_{ext}. We assume leaky integrate-and-fire model neurons with exponentially decaying post-synaptic currents. The dynamics of membrane potential V and synaptic current I_{s} is [57]

$$\begin{aligned} \tau_{\mathrm{m}}\frac{dV_i}{dt} &= -V_i + I_{\mathrm{s},i} \\ \tau_{\mathrm{s}}\frac{dI_{\mathrm{s},i}}{dt} &= -I_{\mathrm{s},i} + \tau_{\mathrm{m}}\sum_{j=1}^{N} w_{ij}\sum_k \delta(t - t_{k,j} - d)\,, \end{aligned} \tag{6.22}$$

where $t_{k,j}$ denotes the k-th spike-time of neuron j, and τ_{m} and τ_{s} are the time constants of membrane and synapse, respectively. The membrane resistance has been absorbed in the definition of the current. Whenever the membrane potential V crosses the threshold θ, the neuron emits a spike and V is reset to the potential V_{r}, where it is clamped for a period of length τ_{r}. Given that all neurons have identical parameters, a diffusion approximation, assuming asynchronous and Poissonian spiking statistics as well as small synaptic couplings, leads to the population-averaged firing rates X_α [57]

$$\begin{aligned} \frac{1}{X_\alpha} &= \tau_{\mathrm{r}} + \tau_{\mathrm{m}}\sqrt{\pi}\int_{(V_{\mathrm{r}}-\mu_\alpha)/\sigma_\alpha+\gamma\sqrt{\tau_{\mathrm{s}}/\tau_{\mathrm{m}}}}^{(\theta-\mu_\alpha)/\sigma_\alpha+\gamma\sqrt{\tau_{\mathrm{s}}/\tau_{\mathrm{m}}}} e^{u^2}\,(1+\mathrm{erf}(u))\,du \\ &=: 1/\Phi_\alpha(X) \end{aligned} \tag{6.23}$$

$$\mu_\alpha = \tau_{\mathrm{m}}\sum_\beta K_{\alpha\beta}w_{\alpha\beta}X_\beta + \tau_{\mathrm{m}}K_{\alpha,\mathrm{ext}}w_{\mathrm{ext}}X_{\mathrm{ext}} \tag{6.24}$$

$$\sigma_\alpha^2 = \tau_{\mathrm{m}}\sum_\beta K_{\alpha\beta}w_{\alpha\beta}^2X_\beta + \tau_{\mathrm{m}}K_{\alpha,\mathrm{ext}}w_{\mathrm{ext}}^2X_{\mathrm{ext}}. \tag{6.25}$$

Here, $\gamma = |\zeta(1/2)|/\sqrt{2}$, with ζ denoting the Riemann zeta function [2]. We find the fixed points of (6.23) by solving the first-order differential equation [166, 188]

$$\tau\frac{dX_\alpha}{dt} = -X_\alpha + \Phi_\alpha(X), \tag{6.26}$$

which constitutes a network of rate units with dimension equal to the number of populations N.

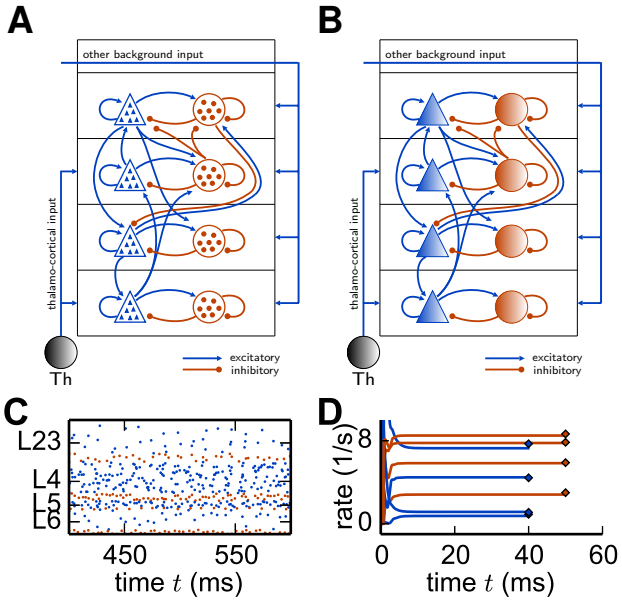

Figure 6.10: **Reduction of spiking microcircuit model to rate dynamics**. **(A)** Sketch of a microcircuit model [149] with excitatory (blue triangles) and inhibitory (red disks) neuron populations, each consisting of a large number of neurons indicated by the small triangles and disks respectively. Arrows between populations indicate the in-degree K. **(B)** Sketch of the corresponding reduced model where each population is replaced by a single rate unit. **(C)** Spiking activity in the different layers. **(D)** Dynamics of the eight units of the rate network (6.26) (curves) and comparison to population-averaged firing rates obtained from direct simulations of the spiking network (diamonds).

Next we apply this framework to a cortical microcircuit model [149] constituting roughly $80,000$ spiking neurons structured into 8 populations across 4 layers $[L23, L4, L5, L6]$, with one excitatory and one inhibitory cell type each (Fig. 6.10). The model exhibits irregular and stationary spiking activity (Fig. 6.10C). Replacing each population by a single rate unit (Fig. 6.10B) results in an eight-dimensional rate network described by differential equation (6.26). If the mean-field approach is valid, the solution of (6.26) converges to a fixed point corresponding to the population-averaged firing rates obtained from direct simulation of the spiking model. Fig. 6.10D shows that the rate network captures the population-averaged firing rates of the spiking model quite well and thus validates the mean-field approach.

The analysis only considers the stationary state of the microcircuit, which can as well be determined using the population-density approach [28]. While the

mean-field approach presented is strictly valid only in the thermodynamic limit, finite-size fluctuations around this state are accessible using the noisy linear-rate model (Subsec. 6.4.3.1) as elaborated by [13] or within the population-density approach [168].

6.5 Discussion

This chapter presents an efficient way to integrate rate-based models in a neuronal network simulator that was originally designed for models with delayed spike-based interactions. The advantage of the latter is a decoupling of neuron dynamics between spike events. This is used by current parallel simulators for large-scale networks of spiking neurons which reduce communication between simulation processes to significantly increase performance and scaling capabilities up to supercomputers [129]. In contrast, rate-based models interact in a continuous way. For delayed interactions, rate dynamics are still decoupled for the minimal delay of the network, such that information can be exchanged on a coarse time-grid. For instantaneous coupling, communication in every time step is required. This is feasible for small networks that can be simulated on small machines and thus require only a small amount of communication. For improved efficiency of simulations of large networks on supercomputers, we implement the waveform-relaxation method which we developed in Sec. 4.2. Furthermore, we investigate several standard methods for the solution of rate-model equations and demonstrate that the scalar exponential Euler method is the best choice in the context of a neuronal network simulator that was originally designed for models with delayed spike-based interactions. Afterwards, we show the applicability of the numerical implementation to a variety of well-known and widely-used rate models.

Our implementation in NEST uses an exponential Euler scheme with a diagonal matrix A (scalar exponential Euler): the leaky dynamics of single neurons are integrated exactly, while the network input to the rate units as well as the additive noise is approximated. The analysis in Subsec. 6.4.1 demonstrates that the scalar exponential Euler method is the most accurate, stable and efficient standard-method for SDEs that is applicable to a distributed spiking simulator. In particular, for all-to-all connected networks of linear rate units the distributed design renders implicit methods less feasible, as the convergence of the involved fixed-point iteration requires small time-steps. For all methods, the computation step size needs to be compared against the time constant τ. Therefore, stable solutions for small values $\tau \ll 1$ may require a decrease of the step size below a default value.

Unlike in our first application of waveform-relaxation methods in a spiking neural network simulator (see Chapter 5) the usage of such a method is only optional for rate-based models. The waveform-relaxation method from Sec. 4.2 can be used for rate-model networks with instantaneous rate connections to improve scalability in large-scale simulations by reducing communication at the cost of additional computations. As a consequence, the optimal method (standard vs. waveform-relaxation) depends on the numbers of compute nodes and virtual processes. In our test case, the use of the waveform-relaxation technique is beneficial for 1024 or more virtual processes. It is therefore recommended to use the iterative scheme for large-scale simulations on supercomputers but to disable it for smaller rate-model simulations on local workstations or laptops. In our implementation this can easily be achieved by the parameter `use_wfr` (see Subsec. 6.2.3 for details). In general, the scalability for simulations of rate models is worse than for spiking network simulations [112] and comparable to simulations with gap junctions (as shown in Chapter 5). This is to be expected, since for rate connections as well as for gap junctions, a large amount of data needs to be communicated compared to a spiking simulation. Future work should assess whether this bottleneck can be overcome by a further optimized communication scheme.

While our implementation uses NEST as a platform, the employed algorithms can be ported to other parallel spiking network simulators. Furthermore, the implementation of the example rate models as templates allows customization to arbitrary gain functions. Researchers can create additional models without in-depth knowledge of simulator specific data structures or numerical methods. In addition, the infrastructure is sufficiently general to allow for extensions to other categories of rate models, as shown explicitly for nonlinear dynamics, multiplicative coupling, and other types of noise. This design enables the usage of the framework for a large body of rate-based network models. The generality of the model equations also supports applications beyond neuronal networks, as e.g. computational gliascience [6] or artificial intelligence [82].

Some design decisions for our implementation come with up- and downsides and may at the present state of knowledge and experience constitute judgment calls: the choice to determine the type of nonlinearity of the recurrent network input with a boolean parameter is based on the assumption that this implementation covers the majority of rate models used in neuroscience today. The solution has an advantage in maintainability, as it results in half as many template instances for a given set of gain functions than the alternative solution discussed in Subsec. 6.2.2. It also avoids the introduction of potentially confusing names of rate models encoding the nature of the nonlinearity. On the downside, models that do actually employ both nonlinearities at once cannot be expressed. Furthermore, a decision that can already be made at the time when the model instance is created is delayed to the simulation phase. The decision to create a separate connection type for

mean-field models of the `siegert` type is made with the goal of avoiding memory overhead. This comes at the price that units of this type cannot be connected to instances of rate models using the generic rate connection. Adapter elements like the `parrot_neuron` (see [109] for a recent application) are one way to overcome this problem. Only the experience of researchers with the present implementation will inform us on whether characteristics and user interface serve the purpose of the community or whether particular decisions need revision.

Mean-field theory has built a bridge between networks of spiking neurons and rate-based units that either represent single neurons or populations [16, 22, 25, 71, 142]. In the latter case, the rate-based approach comes with a considerable reduction of dimensionality (Subsec. 6.4.3.4). Due to a possibly large number of populations, the fixed-point solution of the stationary activity can generally not be determined analytically but can still be found by evolving a pseudo-time dynamics. Within the presented framework, this approach is much faster than the spiking counterpart and thus facilitates the calibration of large-scale spiking network models [166].

Our unifying framework allows researchers to easily switch between rate-based and spiking models in a particular network model requiring only minimal changes to the simulation script. This facilitates an evaluation of the different model types amongst each other and increases reproducibility in the validation of reductions of spiking networks to rate-based models. Furthermore, it is instructive to study whether and how the network dynamics changes with the neuron model [18]. In particular, functional networks being able to perform a given task are typically designed with rate-based units. Their validity can now be evaluated by going from a more abstract rate-based model to a biologically more realistic spiking neuron model. The present implementation in NEST does not allow for interactions between spiking and rate-based units. While this is technically trivial to implement, the proper conversion from spikes to rates and vice versa is a conceptual issue that has to be explored further by theoretical neuroscience.

The presented joined platform for spike-based and rate-based models hopefully triggers new research questions by facilitating collaboration and translation of ideas between scientists working in the two fields. The work presented in this chapter therefore contributes to the unification of both modeling routes in multi-scale approaches, combining large-scale spiking networks with functionally inspired rate-based elements to decipher the dynamics of the brain.

Chapter 7

Conclusions & Outlook

In this thesis we developed waveform-relaxation methods suitable for application in spiking neural network simulators. The main achievements of this thesis are the identification of suitable methods, their efficient implementation in the existing structures of the parallel simulator NEST and their numerical and theoretical analysis.

For the inclusion of gap junctions, i.e. our use case for a waveform-relaxation method applied to ordinary differential equations, we were able to build on a variety of results from a scientifically well researched field. Therefore it is not surprising that our method has common ground with previously developed methods, especially with a class of methods from Bellen and Zerrano (see Sec. 3.1). Nevertheless, the combination of a Runge-Kutta method with adaptive step size control combined with the interpolation on a coarser fixed grid constitutes, to the best of our knowledge, a new approach. This approach is perfectly suited for the application in distributed neural network simulations, as demonstrated by the numerical results in Sec. 5.4.

Beside the mathematical aspects of the inclusion of gap junctions, this part of the thesis contains a lot of conceptional work that can be classified as computer science or more specific, as computational neuroscience. We developed a framework within the existing structures that enables the application of iterative methods in spiking neural network simulators and also enables the communication of arbitrary data that occurs on a regular basis and is not triggered by a spike event. Furthermore, we improved the user interface in order to satisfy the requirements of bidirectional connections.

The inclusion of rate models, i.e. our use case for a waveform-relaxation method applied to stochastic differential equations, raised different challenges. Here we

investigate the choice of the numerical method in detail by analyzing the accuracy and stability properties of a variety of numerical methods for SDEs that are compatible with the general workflow of a spiking neural network simulator. This part, and the entire developed framework, is independent of the use of the waveform-relaxation technique. We do, however, show that large-scale simulations on supercomputers benefit from the use of the waveform-relaxation scheme developed in Sec. 4.2. To this purpose we first prove the convergence of the method in a fixed number of iterations dependent on the step size and the length of the iteration interval, and then demonstrate the benefits by comparing the simulation times of large-scale simulations in the numerical results section. Furthermore, we investigate the benefit of the waveform-relaxation method with respect to the specific rate-unit network with an error bound that is developed in Subsec. 4.2.3.

The release of the developed technology in the NEST simulator makes the results of this thesis visible to the large community of computational neuroscientists. First related publications [97, 169] already indicate that the community will most likely benefit from the new features of the simulator.

In future research, it would be interesting to investigate the performance of more advanced waveform-relaxation schemes in the context of spiking neural network simulators. In this thesis, we restricted the discussion to Jacobi-like waveform-relaxation methods, as this scheme is best suited for the parallel processing that is employed in these simulators. However, other less parallel waveform-relaxation schemes, such as e.g. the Gauss-Seidel scheme, are known to converge significantly faster. It would be interesting to see if a method that represents a compromise between parallel processing and fast convergence can be beneficial in a spiking neural network simulator. Most promising seems a Red-Black Gauss-Seidel scheme, an approach that analyzes the structure of the underlying neural network (subsystems) and identifies groups of neurons (subsystems) that can be processed in parallel, although the underlying scheme in general is Gauss-Seidel. In this context, it needs to be investigated whether the reduction in the number of iterations is significant enough to outdo the additional effort due to pre-processing and increased number of communications per iteration for typical neural network simulations.

List of Figures

List of Tables

List of Algorithms & Scripts

List of Notations

Throughout this thesis, scalars and deterministic vectors are denoted by lower-case letters, and matrices and stochastic elements are denoted by upper-case letters. Subscript $\square_k$ denote time indices, subscript $\square_i$ and $\square_j$ denote component indices and parenthesized superscript $\square^{(m)}$ indicate iteration indices. In addition, the following abbreviations and notations are used across all chapters:

$\mathbb{C}$	the set of complex numbers
$d_{\min}$	minimal network delay
Δt	step size of numerical methods
ΔW	Wiener increment ($\sim \mathcal{N}(0, \Delta t)$)
h	NEST simulator step size
H	Heaviside step function
Hz	hertz
I	identity matrix
ms	millisecond
mV	millivolt
n	total number of time steps
N	total number of components (neurons)
N_{VP}	total number of virtual processes
nS	nanosiemens
pA	picoampere
$\mathbb{R}$	the set of real numbers
$\square^T$	transpose of a matrix or vector
T	interval length of the simulation (biological time simulated)
$\mathcal{T}$	interval length of the waveform relaxation method
T_{sim}	duration of a simulation (simulation time)

Bibliography

[1] L. Abbott, B. DePasquale, and R.-M. Memmesheimer, *Building functional networks of spiking model neurons*, Nat. Neurosci., 19 (2016), pp. 350–355.

[2] M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions: with Formulas, Graphs, and Mathematical Tables*, Dover Publications, New York, 1974.

[3] I. A. Adamu, *Numerical approximation of SDEs and stochastic Swift-Hohenberg equation*, PhD thesis, Heriot-Watt University, 2011.

[4] A. H. Al-Mohy and N. J. Higham, *A new scaling and squaring algorithm for the matrix exponential.*, SIAM J. Matrix Analysis Applications, 31 (2009), pp. 970–989.

[5] S.-I. Amari, *Dynamics of pattern formation in lateral-inhibition type neural fields*, Biol. Cybern., 27 (1977), pp. 77–87.

[6] M. Amiri, F. Bahrami, and M. Janahmadi, *Functional contributions of astrocytes in synchronization of a neuronal network model*, Journal of Theoretical Biology, 292 (2012), pp. 60 – 70.

[7] D. J. Amit and N. Brunel, *Model of global spontaneous activity and local structured activity during delay periods in the cerebral cortex*, Cereb. Cortex, 7 (1997), pp. 237–252.

[8] J. A. Bednar, *Topographica: Building and analyzing map-level simulations from Python, C/C++, MATLAB, NEST, or NEURON components*, Front. Neuroinformatics, 24 (2009).

[9] A. Bellen, Z. Jackiewicz, and M. Zennaro, *Contractivity of waveform relaxation Runge-Kutta iterations and related limit methods for dissipative systems in the maximum norm*, SIAM Journal on Numerical Analysis, 31 (1994), pp. 499–523.

[10] A. Bellen and M. Zennaro, *The use of Runge-Kutta formulae in waveform relaxation methods*, Applied Numerical Mathematics, 11 (1993), pp. 95–114.

[11] M. Bjørhus, *On dynamic iteration for delay differential equations*, BIT Numerical Mathematics, 34 (1994), pp. 325–336.

[12] M. Bjørhus, *A note on the convergence of discretized dynamic iteration*, BIT Numerical Mathematics, 35 (1995), pp. 291–296.

[13] H. Bos, M. Diesmann, and M. Helias, *Identifying anatomical origins of coexisting oscillations in the cortical microcircuit*, PLoS Comput. Biol., 12 (2016), pp. 1–34.

[14] H. Bos, A. Morrison, A. Peyser, J. Hahne, M. Helias, S. Kunkel, et al., *NEST 2.10.0*. `http://dx.doi.org/10.5281/zenodo.44222`, Dec. 2015.

[15] J. M. Bower and D. Beeman, *GENESIS (simulation environment)*, Scholarpedia, 2 (2007), p. 1383.

[16] P. C. Bressloff, *Spatiotemporal dynamics of continuum neural fields*, Journal of Physics A: Mathematical and Theoretical, 45 (2012), p. 033001.

[17] ——, *Path-integral methods for analyzing the effects of fluctuations in stochastic hybrid neural networks*, Journal of Mathematical Neuroscience, 5 (2015).

[18] R. Brette, *Philosophy of the spike: Rate-based vs. spike-based theories of the brain*, Frontiers in Systems Neuroscience, 9 (2015).

[19] R. Brette, M. Rudolph, T. Carnevale, M. Hines, D. Beeman, J. M. Bower, M. Diesmann, Morrison, et al., *Simulation of networks of spiking neurons: A review of tools and strategies*, J. Comput. Neurosci., 23 (2007), pp. 349–398.

[20] N. Brunel, *Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons*, J. Comput. Neurosci., 8 (2000), pp. 183–208.

[21] N. Brunel and V. Hakim, *Fast global oscillations in networks of integrate-and-fire neurons with low firing rates*, Neural Comput., 11 (1999), pp. 1621–1671.

[22] M. A. Buice and C. C. Chow, *Correlations, fluctuations, and stability of a finite-size network of coupled oscillators*, Phys. Rev. E, 76 (2007), p. 031118.

[23] ——, *Dynamic finite size effects in spiking neural networks*, PLoS Comput Biol, 9 (2013), pp. 1–21.

[24] M. A. Buice and J. D. Cowan, *Field-theoretic approach to fluctuation effects in neural networks*, Phys. Rev. E, 75 (2007), p. 051919.

[25] M. A. Buice, J. D. Cowan, and C. C. Chow, *Systematic fluctuation expansion for neural network activity equations*, Neural Comput., 22 (2010), pp. 377–426.

[26] K. Burrage, *Parallel methods for initial value problems*, Applied Numerical Mathematics, 11 (1993), pp. 5–25.

[27] G. D. Byrne and A. C. Hindmarsh, *PVODE, an ODE solver for parallel computers*, International Journal of High Performance Computing Applications, 13 (1999), pp. 354–365.

[28] N. Cain, R. Iyer, C. Koch, and S. Mihalas, *The computational properties of a simplified cortical column model*, PLoS Comput. Biol., 12 (2016), p. e1005045.

[29] T. Carnevale and M. Hines, *The NEURON Book*, Cambridge University Press, Cambridge, 2006.

[30] A. Casaleggio, M. L. Hines, and M. Migliore, *Computational model of erratic arrhythmias in a cardiac cell network: The role of gap junctions*, PloS one, 9 (2014), p. e100288.

[31] R. Chaudhuri, K. Knoblauch, M.-A. Gariel, H. Kennedy, and X.-J. Wang, *A large-scale circuit mechanism for hierarchical dynamical processing in the primate cortex*, Neuron, 88 (2015), pp. 419–431.

[32] A. Cichocki, R. Zdunek, A. H. Phan, and S.-i. Amari, *Nonnegative matrix and tensor factorizations: Applications to exploratory multi-way data analysis and blind source separation*, John Wiley & Sons, Chichester, 2009.

[33] S. D. Cohen and A. C. Hindmarsh, *CVODE, a stiff/nonstiff ODE solver in C*, Computers in Physics, 10 (1996), pp. 138–143.

[34] B. W. Connors and M. A. Long, *Electrical synapses in the mammalian brain*, Annu. Rev. Neurosci., 27 (2004), pp. 393–418.

[35] S. Coombes, *Waves, bumps, and patterns in neural field theories*, Biol. Cybern., 93 (2005), pp. 91–108.

[36] S. Coombes and M. Zachariou, *Gap junctions and emergent rhythms*, in Coherent Behavior in Neuronal Networks, K. Josic, J. Rubin, M. Matias, and R. Romo, eds., vol. 3 of Springer Series in Computational Neuroscience, Springer New York, 2009, pp. 77–94.

[37] D. Dahmen, H. Bos, and M. Helias, *Correlated fluctuations in strongly coupled binary networks beyond equilibrium*, Phys. Rev. X, 6 (2016), p. 031024.

[38] A. Davison, D. Brüderle, J. Eppler, J. Kremkow, E. Muller, D. Pecevski, L. Perrinet, and P. Yger, *PyNN: A common interface for neuronal network simulators*, Front. Neuroinformatics, 2 (2008).

[39] M. de Kamps, *A generic approach to solving jump diffusion equations with applications to neural populations*, ArXiv e-prints, 1309.1654v2 [q-bio.NC] (2013).

[40] M. de Kamps, V. Baier, J. Drever, M. Dietz, L. Mösenlechner, and F. van der Felde, *The state of miind*, Neural Networks, 21 (2008), pp. 1164–1181.

[41] E. Deadman, N. J. Higham, and R. Ralha, *Blocked Schur algorithms for computing the matrix square root*, in PARA, P. Manninen and P. Öster, eds., vol. 7782 of Lecture Notes in Computer Science, Heidelberg, 2012, Springer, pp. 171–182.

[42] G. Deco and V. K. Jirsa, *Ongoing cortical activity at rest: Criticality, multistability, and ghost attractors*, J. Neurosci., 32 (2012), pp. 3366–3375.

[43] G. Deco, V. K. Jirsa, and A. R. McIntosh, *Emerging concepts for the dynamical organization of resting-state activity in the brain*, Nat. Rev. Neurosci., 12 (2011), pp. 43–56.

[44] G. Deco, V. K. Jirsa, P. A. Robinson, M. Breakspear, and K. Friston, *The dynamic brain: From spiking neurons to neural masses and cortical fields*, PLoS Comput. Biol., 4 (2008), p. e1000092.

[45] E. Dere and A. Zlomuzica, *The role of gap junctions in the brain in health and disease.*, Neurosci. Biobehav. Rev., 36 (2011), pp. 206–217.

[46] A. Destexhe and D. Paré, *Impact of network activity on the integrative properties of neocortical pyramidal neurons in vivo*, J. Neurophysiol., 81 (1999), pp. 1531–1547.

[47] M. Djurfeldt, A. P. Davison, and J. M. Eppler, *Efficient generation of connectivity in neuronal networks from simulator-independent descriptions*, Front. Neuroinformatics, 8 (2014).

[48] M. Djurfeldt, J. Hjorth, J. M. Eppler, N. Dudani, M. Helias, T. C. Potjans, U. S. Bhalla, M. Diesmann, J. Hellgren Kotaleski, and O. Ekeberg, *Run-time interoperability between neuronal network simulators based on the MUSIC framework*, Neuroinformatics, 8 (2010), pp. 43–60.

[49] M. Djurfeldt, M. Lundqvist, C. Johansson, M. Rehn, O. Ekeberg, and A. Lansner, *Brain-scale simulation of the neocortex on the IBM Blue Gene/L supercomputer*, IBM Journal of Research and Development, 52 (2008), pp. 31–41.

[50] A. S. Ecker, P. Berens, G. A. Keliris, M. Bethge, and N. K. Logothetis, *Decorrelated neuronal firing in cortical microcircuits*, Science, 327 (2010), pp. 584–587.

[51] J. M. Eppler, M. Helias, E. Muller, M. Diesmann, and M. Gewaltig, *PyNEST: A convenient interface to the NEST simulator*, Front. Neuroinformatics, 2 (2009).

[52] J. M. Eppler, R. Pauli, A. Peyser, T. Ippen, A. Morrison, J. Senk, W. Schenck, et al., *NEST 2.8.0*. `http://dx.doi.org/10.5281/zenodo.32969`, Sept. 2015.

[53] Z. Fan, *Discrete time waveform relaxation method for stochastic delay differential equations*, Applied Mathematics and Computation, 217 (2010), pp. 3903–3909.

[54] ——, *Waveform relaxation method for stochastic differential equations with constant delay*, Applied Numerical Mathematics, 61 (2011), pp. 229–240.

[55] ——, *SOR waveform relaxation methods for stochastic differential equations*, Appl. Math. Comput., 219 (2013), pp. 4992–5003.

[56] J. A. Feldman and D. H. Ballard, *Connectionist models and their properties*, Cognitive science, 6 (1982), pp. 205–254.

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

[58] N. Fourcaud-Trocmé, D. Hansel, C. van Vreeswijk, and N. Brunel, *How spike generation mechanisms determine the neuronal response to fluctuating inputs*, J. Neurosci., 23 (2003), pp. 11628–11640.

[59] S. Furber, D. Lester, L. Plana, J. Garside, E. Painkras, S. Temple, and A. Brown, *Overview of the spinnaker system architecture*, IEEE Trans. Comp., 62 (2013), pp. 2454–2467.

[60] G. Gancarz and S. Grossberg, *A neural model of the saccade generator in the reticular formation*, IEEE Trans. Neural Netw., 11 (1998), pp. 1159–1174.

[61] C. W. Gardiner, *Handbook of Stochastic Methods for Physics, Chemistry and the Natural Sciences*, Springer Series in Synergetics, Springer, Berlin, 3rd ed., 2004.

[62] L. Gentet, M. Avermann, F. Matyas, J. F. Staiger, and C. C. Petersen, *Membrane potential dynamics of GABAergic neurons in the barrel cortex of behaving mice*, Neuron, 65 (2010), pp. 422–435.

[63] W. Gerstner, *Time structure of the activity in neural network models*, Phys. Rev. E, 51 (1995), pp. 738–758.

[64] ——, *Population dynamics of spiking neurons: fast transients, asynchronous states, and locking*, Neural Comput., 12 (2000), pp. 43–89.

[65] W. Gerstner and W. Kistler, *Spiking Neuron Models: Single Neurons, Populations, Plasticity*, Cambridge University Press, 2002.

[66] I. Ginzburg and H. Sompolinsky, *Theory of correlations in stochastic neural networks*, Phys. Rev. E, 50 (1994), pp. 3171–3191.

[67] S. Goedeke, J. Schuecker, and M. Helias, *Noise dynamically suppresses chaos in neural networks*, ArXiv e-prints, 1603.01880v2 [q-bio.NC] (2016).

[68] D. Goodman and R. Brette, *Brian simulator*, Scholarpedia, 8 (2013), p. 10883.

[69] J. S. Griffith and G. Horn, *An analysis of spontaneous impulse activity of units in the striate cortex of unrestrained cats*, J. Physiol. (Lond.), 186 (1966), pp. 516–534.

[70] S. Grossberg, *Contour enhancement, short term memory, and constancies in reverberating neural networks*, Stud. Appl. Math., 52 (1973), pp. 213–257.

[71] D. Grytskyy, T. Tetzlaff, M. Diesmann, and M. Helias, *A unified view on weakly correlated recurrent networks*, Front. Comput. Neurosci., 7 (2013).

[72] J. Hahne, D. Dahmen, J. Schuecker, A. Frommer, M. Bolten, M. Helias, and M. Diesmann, *Integration of continuous-time dynamics in a spiking neural network simulator*, Front. Neuroinformatics, 11 (2017).

[73] J. Hahne, M. Helias, S. Kunkel, J. Igarashi, M. Bolten, A. Frommer, and M. Diesmann, *A unified framework for spiking and gap-junction interactions in distributed neuronal network simulations.*, Front. Neuroinformatics, 9 (2015).

[74] J. Hahne, M. Helias, S. Kunkel, J. Igarashi, I. Kitayama, B. Wylie, M. Bolten, A. Frommer, and M. Diesmann, *Including gap junctions into distributed neuronal network simulations*, in Brain-Inspired Computing: Second International Workshop, BrainComp 2015, Cetraro, Italy, July 6-10, 2015, Revised Selected Papers, K. Amunts, L. Grandinetti, T. Lippert, and N. Petkov, eds., Springer International Publishing, Cham, 2016, pp. 43–57.

[75] E. Hairer, S. Nørsett, and G. Wanner, *Solving Ordinary Differential Equations I: Nonstiff problems*, Springer, Berlin, 2nd ed., 2000.

[76] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II*, Springer, Berlin, 1991.

[77] D. Hansel and G. Mato, *Asynchronous states and the emergence of synchrony in large networks of interacting excitatory and inhibitory neurons*, Neural Comput., 15 (2003), pp. 1–56.

[78] D. Hansel, G. Mato, and P. Benjamin, *The role of intrinsic cell properties in synchrony of neurons interacting via electrical synapses*, in Phase Response Curves in Neuroscience: Theory, Experiment, and Analysis, N. W. Schultheiss, A. A. Prinz, and R. J. Butera, eds., vol. 6 of Springer Series in Computational Neuroscience, Springer, 1st ed., 2012, ch. 15, pp. 361–398.

[79] D. Hansel, G. Mato, C. Meunier, and L. Neltner, *On numerical simulations of integrate-and-fire neural networks*, Neural Comput., 10 (1998), pp. 467–483.

[80] D. Hansel and H. Sompolinsky, *Modeling feature selectivity in local cortical circuits*, in Methods in Neuronal Modeling, C. Koch and I. Segev, eds., MIT Press, Cambridge, Massachusetts, 2nd ed., 1998, pp. 499–567.

[81] A. Hanuschkin, S. Kunkel, M. Helias, A. Morrison, and M. Diesmann, *A general and efficient method for incorporating precise spike times in globally time-driven simulations*, Front. Neuroinformatics, 4 (2010).

[82] S. S. Haykin, *Neural Networks and Learning Machines*, Prentice Hall, New York, 3rd ed., 2009.

[83] M. Helias, S. Kunkel, G. Masumoto, J. Igarashi, J. M. Eppler, S. Ishii, T. Fukai, A. Morrison, and M. Diesmann, *Supercomputers ready for use as discovery machines for neuroscience*, Front. Neuroinformatics, 6 (2012).

[84] M. Helias, S. Rotter, M.-O. Gewaltig, and M. Diesmann, *Structural plasticity controlled by calcium based correlation detection*, Front. Comput. Neurosci., 2 (2008).

[85] M. Helias, T. Tetzlaff, and M. Diesmann, *Echoes in correlated neural systems*, New J. Phys., 15 (2013), p. 023002.

[86] ——, *The correlation structure of local cortical networks intrinsically results from recurrent dynamics*, PLoS Comput. Biol., 10 (2014), p. e1003428.

[87] S. Henker, J. Partzsch, and R. Schüffny, *Accuracy evaluation of numerical methods used in state-of-the-art simulators for spiking neural networks*, J. Comput. Neurosci., 32 (2012), pp. 309–326.

[88] S. Herculano-Houzel, *The human brain in numbers: A linearly scaled-up primate brain*, Front. Hum. Neurosci., 3 (2009).

[89] J. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the Theory of Neural Computation*, Perseus Books, 1991.

[90] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward, *SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers.*, ACM Trans. Math. Softw., 31 (2005), pp. 363–396.

[91] M. Hines, H. Eichner, and F. Schürmann, *Neuron splitting in compute-bound parallel network simulations enables runtime scaling with twice as many processors*, J Comput Neurosci., 25 (2008), pp. 203–210.

[92] M. Hines, S. Kumar, and F. Schürmann, *Comparison of neuronal spike exchange methods on a Blue Gene/P supercomputer*, Front. Comput. Neurosci., 5 (2011). 10.3389/fncom.2011.00049.

[93] M. L. Hines, H. Markram, and F. Schürmann, *Fully implicit parallel simulation of single neurons*, J. Comput. Neurosci., 25 (2008), pp. 439–448.

[94] S. Hormuzdi, M. Filippov, G. Mitropoulou, H. Monyer, and R. Bruzzone, *Electrical synapses: A dynamic signaling system that shapes the activity of neuronal networks*, Biochim Biophys Acta, 1662 (2004), pp. 113–137.

[95] K. J. in't Hout, *On the convergence of waveform relaxation methods for stiff nonlinear ordinary differential equations*, Applied Numerical Mathematics, 18 (1995), pp. 175–190.

[96] J. Jordan, T. Ippen, M. Helias, I. Kitayama, M. Sato, J. Igarashi, M. Diesmann, and S. Kunkel, *Extremely scalable spiking neuronal network simulation code: From laptops to exascale computers*, Front. Neuroinformatics, 12 (2018).

[97] J. Jordan, P. Weidel, and A. Morrison, *Closing the loop between neural network simulators and the OpenAI Gym*, ArXiv e-prints, 1709.05650 [q-bio.NC] (2017).

[98] Jülich Supercomputing Centre, *JUQUEEN: IBM Blue Gene/Q® supercomputer system at the Jülich Supercomputing Centre*, Journal of large-scale research facilities, 1 (2015).

[99] R. E. Kaas, U. Eden, and E. N. Brown, *Analysis of Neural Data*, Springer Series in Statistics, Springer, 2014.

[100] J. Karbowski and N. Kopell, *Multispikes and synchronization in a large neural network with temporal delays*, Neural Comput., 12 (2000), pp. 1573–1606.

[101] C. T. Kelley, *Iterative Methods for Linear and Nonlinear Equations*, no. 16 in Frontiers in Applied Mathematics, SIAM, 1995.

[102] Z. P. Kilpatrick, *Wilson-Cowan model*, in Encyclopedia of Computational Neuroscience, D. Jaeger and R. Jung, eds., Springer, New York, 2015, pp. 3159–3163.

[103] P. E. Kloeden and E. Platen, *Numerical Solution of Stochastic Differential Equations*, Springer, Berlin, 1992.

[104] B. W. Knight, *Dynamics of encoding in a population of neurons*, J. Gen. Physiol., 59 (1972), pp. 734–766.

[105] K. W. Koch and J. M. Fuster, *Unit activity in monkey parietal cortex related to haptic perception and temporary memory*, Exp. Brain Res., 76 (1989), pp. 292–306.

[106] Y. Komori and K. Burrage, *A stochastic exponential Euler scheme for simulation of stiff biochemical reaction systems*, BIT Numerical Mathematics, 54 (2014), pp. 1067–1085.

[107] B. Kriener, H. Enger, T. Tetzlaff, H. E. Plesser, M.-O. Gewaltig, and G. T. Einevoll, *Dynamics of self-sustained asynchronous-irregular activity in random networks of spiking neurons with strong synapses.*, Front. Comput. Neurosci., 8 (2014).

[108] S. Kumar, P. Heidelberger, D. Chen, and M. Hines, *Optimization of applications with non-blocking neighborhood collectives via multisends on the blue gene/p supercomputer*, IPDPD, (2010), pp. 1–11.

[109] S. Kunkel, M. Diesmann, and A. Morrison, *Limits to the development of feed-forward structures in large recurrent neuronal networks*, Front. Comput. Neurosci., 4 (2011).

[110] S. Kunkel, A. Morrison, P. Weidel, J. M. Eppler, A. Sinha, W. Schenck, M. Schmidt, S. B. Vennemo, J. Jordan, A. Peyser, D. Plotnikov, S. Graber, T. Fardet, D. Terhorst, H. Mørk, G. Trensch, A. Seeholzer, R. Deepu, J. Hahne, et al., *NEST 2.12.0*. `https://doi.org/10.5281/zenodo.259534`, Mar. 2017.

[111] S. Kunkel, T. C. Potjans, J. M. Eppler, H. E. Plesser, A. Morrison, and M. Diesmann, *Meeting the memory challenges of brain-scale simulation*, Front. Neuroinformatics, 5 (2012).

[112] S. Kunkel, M. Schmidt, J. M. Eppler, G. Masumoto, J. Igarashi, S. Ishii, T. Fukai, A. Morrison, M. Diesmann, and M. Helias, *Spiking network simulation code for petascale computers*, Front. Neuroinformatics, 8 (2014).

[113] E. Lelarasmee, *The waveform relaxation method for time domain analysis of large scale integrated circuits: theory and applications*, Memorandum No. UCB/ERL M82/40, (1982).

[114] E. Lelarasmee, A. E. Ruehli, and A. L. Sangiovanni-Vincentelli, *The waveform relaxation method for time-domain analysis of large scale integrated circuits.*, IEEE Trans. on CAD of Integrated Circuits and Systems, 1 (1982), pp. 131–145.

[115] E. Lindelöf, *Sur l'application de la méthode des approximations successives aux équations différentielles ordinaires du premier ordre*, Comptes rendus hebdomadaires des séances de l'Académie des sciences, 116 (1894), pp. 454–457.

[116] B. Lindner, B. Doiron, and A. Longtin, *Theory of oscillatory firing induced by spatially correlated noise and delayed inhibitory feedback*, Phys. Rev. E, 72 (2005), p. 061919.

[117] R. Malach, Y. Amir, M. Harel, and A. Grinvald, *Relationship between intrinsic connections and functional architecture revealed by optical imaging and in vivo targeted biocytin injections in primate striate cortex*, Proc. Natl. Acad. Sci. USA, 90 (1993), pp. 10469–10473.

[118] J. G. Mancilla, T. J. Lewis, D. J. Pinto, J. Rinzel, and B. W. Connors, *Synchronization of electrically coupled pairs of inhibitory interneurons in neocortex*, J. Neurosci., 27 (2007), pp. 2058–2073.

[119] H. Markram, E. Muller, S. Ramaswamy, M. W. Reimann, M. Abdellah, C. A. Sanchez, A. Ailamaki, L. Alonso-Nanclares, N. Antille, S. Arsever, et al., *Reconstruction and simulation of neocortical microcircuitry*, Cell, 163 (2015), pp. 456–492.

[120] M. Mattia and P. Del Guidice, *Population dynamics of interacting spiking neurons*, Phys. Rev. E, 66 (2002), p. 051917.

[121] ——, *Finite-size dynamics of inhibitory and excitatory interacting spiking neurons*, Phys. Rev. E, 70 (2004), p. 052903.

[122] J. L. McClelland and D. E. Rumelhart, *An interactive activation model of context effects in letter perception: I. an account of basic findings.*, Psychological review, 88 (1981), pp. 375–407.

[123] ——, *Explorations in parallel distributed processing: A handbook of models, programs, and exercises*, MIT press, Cambridge, 1989.

[124] Message Passing Interface Forum, *MPI: A message-passing interface standard, version 2.2*, tech. rep., MPI Forum, 2009.

[125] C. Meyer and C. van Vreeswijk, *Temporal correlations in stochastic networks of spiking neurons*, Neural Comput., 14 (2002), pp. 369–404.

[126] H. Miyazaki, Y. Kusano, N. Shinjou, S. Fumiyoshi, M. Yokokawa, and T. Watanabe, *Overview of the K computer System*, Fujitsu Scientific and Technical Journal, 48 (2012), pp. 255–265.

[127] E. Montbrió, D. Pazó, and A. Roxin, *Macroscopic description for networks of spiking neurons*, Phys. Rev. X, 5 (2015), p. 021028.

[128] A. Morrison and M. Diesmann, *Maintaining causality in discrete time neuronal network simulations*, in Lectures in Supercomputational Neurosciences: Dynamics in Complex Brain Networks, P. b. Graben, C. Zhou, M. Thiel, and J. Kurths, eds., Springer, Berlin, Heidelberg, 2008, pp. 267–278.

[129] A. Morrison, C. Mehring, T. Geisel, A. Aertsen, and M. Diesmann, *Advancing the boundaries of high connectivity network simulation with distributed computing*, Neural Comput., 17 (2005), pp. 1776–1801.

[130] A. Morrison, S. Straube, H. E. Plesser, and M. Diesmann, *Exact subthreshold integration with continuous spike times in discrete time neural network simulations*, Neural Comput., 19 (2007), pp. 47–79.

[131] O. Nevanlinna, *Remarks on Picard-Lindelöf iteration, Part I*, BIT Numerical Mathematics, 29 (1989), pp. 328–346.

[132] ——, *Remarks on Picard-Lindelöf iteration, Part II*, BIT Numerical Mathematics, 29 (1989), pp. 535–562.

[133] E. J. Nichols and A. Hutt, *Neural field simulator: Two-dimensional spatio-temporal dynamics involving finite transmission speed*, Front. Neuroinformatics, 9 (2015).

[134] M. Ohbayashi, K. Ohki, and Y. Miyashita, *Conversion of working memory to motor sequence in the monkey premotor cortex*, Science, 301 (2003), pp. 233–236.

[135] B. Øksendal, *Stochastic differential equations*, Springer, Berlin, 6th ed., 2003.

[136] M. Okun and I. Lampl, *Instantaneous correlation of excitation and inhibition during ongoing and sensory-evoked activities*, Nat. Neurosci., 11 (2008), pp. 535–537.

[137] OpenMP Architecture Review Board, *OpenMP application program interface*, specification, OpenMP ARB, `http://www.openmp.org/mp-documents/spec30.pdf`, 2008.

[138] R. C. O'Reilly, *Comparison of neural network simulators.* `https://grey.colorado.edu/emergent/index.php/Comparison_of_Neural_Network_Simulators`, 2014. Accessed: 2016-10-14.

[139] R. C. O'Reilly, Y. Munakata, M. J. Frank, T. E. Hazy, and Contributors, *Computational Cognitive Neuroscience*, Wiki Book, 1st Edition, URL: `http://ccnbook.colorado.edu`, 2012.

[140] R. C. O'Reilly, Y. Munakata, and J. L. McClelland, *Computational Explorations in Cognitive Neuroscience: Understanding the Mind by Simulating the Brain*, MIT Press, Cambridge, 1st ed., 2000.

[141] S. Ostojic, *Two types of asynchronous activity in networks of excitatory and inhibitory spiking neurons*, Nat. Neurosci., 17 (2014), pp. 594–600.

[142] S. Ostojic and N. Brunel, *From spiking neuron models to linear-nonlinear models*, PLoS Comput. Biol., 7 (2011), p. e1001056.

[143] N. Paulauskas, H. Pranevicius, J. Mockus, and F. F. Bukauskas, *Stochastic 16-state model of voltage gating of gap-junction channels enclosing fast and slow gates*, Biophysical journal, 102 (2012), pp. 2471–2480.

[144] V. Pernice, B. Staude, S. Cardanobile, and S. Rotter, *How structure determines correlations in neuronal networks*, PLoS Comput. Biol., 7 (2011), p. e1002059.

[145] A. Peyser, A. Sinha, S. B. Vennemo, T. Ippen, J. Jordan, S. Graber, A. Morrison, G. Trensch, T. Fardet, H. Mørk, J. Hahne, et al., *NEST 2.14.0*. `https://doi.org/10.5281/zenodo.882971`, Oct. 2017.

[146] B. Pfeuty, G. Mato, D. Golomb, and D. Hansel, *Electrical synapses and synchrony: The role of intrinsic currents*, J. Neurosci., 23 (2003), pp. 6280–6294.

[147] H. E. Plesser, J. M. Eppler, A. Morrison, M. Diesmann, and M.-O. Gewaltig, *Efficient parallel simulation of large-scale neuronal networks on clusters of multiprocessor computers*, in Euro-Par 2007: Parallel Processing, A.-M. Kermarrec, L. Bougé, and T. Priol, eds., vol. 4641 of Lecture Notes in Computer Science, Berlin, 2007, Springer, pp. 672–681.

[148] D. Plotnikov, I. Blundell, T. Ippen, J. M. Eppler, A. Morrison, and B. Rumpe, *NESTML: a modeling language for spiking neurons*, in Modellierung 2016 Conference, vol. 254 of LNI, Bonn, 2016, Bonner Köllen Verlag, pp. 93–108.

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

[150] K. Rajan and L. F. Abbott, *Eigenvalue spectra of random matrices for neural networks*, Phys. Rev. Lett., 97 (2006), p. 188104.

[151] A. Renart, J. De La Rocha, P. Bartho, L. Hollender, N. Parga, A. Reyes, and K. D. Harris, *The asynchronous state in cortical circuits*, Science, 327 (2010), pp. 587–590.

[152] H. Risken, *The Fokker-Planck Equation*, Springer Verlag Berlin, 1996.

[153] S. Rotter and M. Diesmann, *Exact digital simulation of time-invariant linear systems with applications to neuronal modeling*, Biol. Cybern., 81 (1999), pp. 381–402.

[154] N. P. Rougier and J. Fix, *Dana: Distributed numerical and adaptive modelling framework*, Network: Computation in Neural Systems, 23 (2012), pp. 237–253.

[155] A. Roxin, N. Brunel, and D. Hansel, *The role of delays in shaping spatio-temporal dynamics of neuronal activity in large networks*, Phys. Rev. Lett., 94 (2005), p. 238103.

[156] ——, *Rate models with delays and the dynamics of large networks of spiking neurons*, Progress of Theoretical Physics Supplement, 161 (2006), pp. 68–85.

[157] A. Roxin, N. Brunel, D. Hansel, G. Mongillo, and C. van Vreeswijk, *On the distribution of firing rates in networks of cortical neurons*, J. Neurosci., 31 (2011), pp. 16217–16226.

[158] D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, *Parallel Distributed Processing, Explorations in the Microstructure of Cognition: Foundations*, vol. 1, MIT Press, Cambridge, Massachusetts, 1986.

[159] S. Sadeh and S. Rotter, *Orientation selectivity in inhibition-dominated networks of spiking neurons: Effect of single neuron properties and network dynamics*, PLoS Comput Biol, 11 (2015), p. e1004045.

[160] J. Sand and K. Burrage, *A Jacobi waveform relaxation method for ODEs*, SIAM J. Scientific Computing, 20 (1998), pp. 534–552.

[161] P. Sanz Leon, S. Knock, M. Woodman, L. Domide, J. Mersmann, A. McIntosh, and V. Jirsa, *The virtual brain: A simulator of primate brain network dynamics*, Front. Neuroinformatics, 7 (2013).

[162] M. Schmidt, R. Bakker, M. Diesmann, and S. J. van Albada, *Full-density multi-scale account of structure and dynamics of macaque visual cortex*, ArXiv e-prints, 1511.09364v3 [q-bio.NC] (2016).

[163] G. Schöner, J. Spencer, and D. Group, *Dynamic Thinking: A Primer on Dynamic Field Theory*, Oxford Series in Developmental Cognitive Neuroscience, Oxford University Press, Oxford, 2015.

[164] J. Schuecker, M. Diesmann, and M. Helias, *Modulated escape from a metastable state driven by colored noise*, Phys. Rev. E, 92 (2015), p. 052119.

[165] J. Schuecker, S. Goedeke, D. Dahmen, and M. Helias, *Functional methods for disordered neural networks*, ArXiv e-prints, 1605.06758v2 [cond-mat.dis-nn] (2016).

[166] J. Schuecker, M. Schmidt, S. J. van Albada, M. Diesmann, and M. Helias, *Fundamental activity constraints lead to specific interpretations of the connectome*, PLoS Comput. Biol., 13 (2017), p. e1005179.

[167] H. Schurz and K. R. Schneider, *Waveform relaxation methods for stochastic differential equations*, International Journal of Numerical Analysis and Modeling, 3 (2005), pp. 232–254.

[168] T. Schwalger, M. Deger, and W. Gerstner, *Towards a theory of cortical columns: From spiking neurons to interacting neural populations of finite size*, PLOS Computational Biology, 13 (2017), pp. 1–63.

[169] M. Senden, J. Schuecker, J. Hahne, M. Diesmann, and R. Goebel, *[Re] A neural model of the saccade generator in the reticular formation*, ReScience, 4 (2018).

[170] M. N. Shadlen and W. T. Newsome, *The variable discharge of cortical neurons: Implications for connectivity, computation, and information coding*, J. Neurosci., 18 (1998), pp. 3870–3896.

[171] I. Shoji, *A note on convergence rate of a linearization method for the discretization of stochastic differential equations*, Communications in Nonlinear Science and Numerical Simulation, 16 (2011), pp. 2667–2671.

[172] A. J. Siegert, *On the first passage time probability problem*, Phys. Rev., 81 (1951), pp. 617–623.

[173] W. R. Softky and C. Koch, *The highly irregular firing of cortical cells is inconsistent with temporal integration of random EPSPs*, J. Neurosci., 13 (1993), pp. 334–350.

[174] H. Sompolinsky, A. Crisanti, and H. J. Sommers, *Chaos in random neural networks*, Phys. Rev. Lett., 61 (1988), pp. 259–262.

[175] M. Stern, H. Sompolinsky, and L. F. Abbott, *Dynamics of random neural networks with bistable units*, Phys. Rev. E, 90 (2014), p. 062710.

[176] J. Stoer and R. Bulirsch, *Introduction to Numerical Analysis*, Springer, New York, 1st ed., 1980.

[177] T. Tetzlaff, M. Helias, G. Einevoll, and M. Diesmann, *Decorrelation of neural-network activity by inhibitory feedback*, PLoS Comput. Biol., 8 (2012), p. e1002596.

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

[179] S. J. van Albada, M. Helias, and M. Diesmann, *Scalability of asynchronous networks is limited by one-to-one mapping between effective connectivity and correlations*, PLoS Comput. Biol., 11 (2015), p. e1004490.

[180] C. van Vreeswijk and H. Sompolinsky, *Chaos in neuronal networks with balanced excitatory and inhibitory activity*, Science, 274 (1996), pp. 1724–1726.

[181] K. Vervaeke, A. Lőrincz, Z. Nusser, and R. A. Silver, *Gap junctions compensate for sublinear dendritic integration in an inhibitory network*, Science, 335 (2012), pp. 1624–1628.

[182] N. Voges and L. U. Perrinet, *Complex dynamics in recurrent cortical networks based on spatially realistic connectivities*, Front. Comput. Neurosci., 6 (2012).

[183] N. Voges, A. Schüz, A. Aertsen, and S. Rotter, *A modeler's view on the spatial structure of intrinsic horizontal connectivity in the neocortex*, Prog. Neurobiol., 92 (2010), pp. 277–292.

[184] X.-J. Wang and G. Buzsáki, *Gamma oscillation by synaptic inhibition in a hippocampal interneuronal network model*, J. Neurosci., 16 (1996), pp. 6402–6413.

[185] A. Weitzenfeld, M. A. Arbib, and A. Alexander, *The neural simulation language: A system for brain modeling*, MIT Press, Cambridge, 2002.

[186] H. R. Wilson and J. D. Cowan, *Excitatory and inhibitory interactions in localized populations of model neurons*, Biophys. J., 12 (1972), pp. 1–24.

[187] ——, *A mathematical theory of the functional dynamics of cortical and thalamic nervous tissue*, Kybernetik, 13 (1973), pp. 55–80.

[188] K.-F. Wong and X.-J. Wang, *A recurrent network mechanism of time integration in perceptual decisions*, J. Neurosci., 26 (2006), pp. 1314–1328.

[189] P. Yger, S. El Boustani, A. Destexhe, and Y. Frégnac, *Topologically invariant macroscopic statistics in balanced networks of conductance-based integrate-and-fire neurons*, J. Comput. Neurosci., 31 (2011), pp. 229–245.