LETTER Communicated by Alain Destexhe

Exact Subthreshold Integration with Continuous Spike Times in Discrete-Time Neural Network Simulations

Abigail Morrison
abigail@biologie.uni-freiburg.de
Computational Neurophysics, Institute of Biology III, and Bernstein Center for Computational Neuroscience, Albert-Ludwigs-University, 79104 Freiburg, Germany

Sirko Straube
sirko.straube@biologie.uni-freiburg.de
Computational Neurophysics, Institute of Biology III, Albert-Ludwigs-University, 79104 Freiburg, Germany

Hans Ekkehard Plesser
hans.ekkehard.plesser@umb.no
Department of Mathematical Sciences and Technology, Norwegian University of Life Sciences, N-1432 Ås, Norway

Markus Diesmann
diesmann@biologie.uni-freiburg.de
Computational Neurophysics, Institute of Biology III, and Bernstein Center for Computational Neuroscience, Albert-Ludwigs-University, 79104 Freiburg, Germany

Very large networks of spiking neurons can be simulated efficiently in parallel under the constraint that spike times are bound to an equidistant time grid. Within this scheme, the subthreshold dynamics of a wide class of integrate-and-fire-type neuron models can be integrated exactly from one grid point to the next. However, the loss in accuracy caused by restricting spike times to the grid can have undesirable consequences, which has led to interest in interpolating spike times between the grid points to retrieve an adequate representation of network dynamics. We demonstrate that the exact integration scheme can be combined naturally with off-grid spike events found by interpolation. We show that by exploiting the existence of a minimal synaptic propagation delay, the need for a central event queue is removed, so that the precision of event-driven simulation on the level of single neurons is combined with the efficiency of time-driven global scheduling. Further, for neuron models with linear subthreshold dynamics, even local event queuing can be avoided, resulting in much greater efficiency on the single-neuron level. These ideas are exemplified by two implementations of a widely used neuron model. We present a measure for the efficiency of network simulations in terms

of their integration error and show that for a wide range of input spike rates, the novel techniques we present are both more accurate and faster than standard techniques.

1 Introduction

A major problem in the simulation of cortical neural networks has been their high connectivity. With each neuron receiving input from the order of 10^4 other neurons, simulations are demanding in terms of memory as well as simulation time requirements. Techniques to study such highly connected systems of 10^5 and more neurons, using distributed computing, are now available (see, e.g., Hammarlund & Ekeberg, 1998; Harris et al., 2003; Morrison, Mehring, Geisel, Aertsen, & Diesmann, 2005). The question remains as to whether a time-driven or event-driven simulation algorithm (Fujimoto, 2000; Zeigler, Praehofer, & Kim, 2000; Sloot, Kaandorp, Hoekstra, & Overeinder, 1999; Ferscha, 1996) should be used. At first glance, the choice of event-driven algorithms seems natural, because a neuron can be described as emitting and absorbing point events (spikes) in continuous time. In fact, for neuron models with linear subthreshold dynamics and postsynaptic potentials without rise time, highly efficient algorithms exist (see, e.g., Mattia & Del Giudice, 2000). These exploit the fact that threshold crossings can occur only at the impact times of excitatory events. If more general types of neuron models are considered, the global algorithmic framework becomes much more complicated. For example, each neuron may be required to "look ahead" to determine when it will fire in the absence of new events. The global algorithm then either updates the neuron with the shortest latency or delivers the event with the most imminent arrival time (whichever is shorter) and revises the latency calculations for the neurons receiving the event. (See Marian, Reilly, & Mackey, 2002; Makino, 2003; Rochel & Martinez, 2003; Lytton & Hines, 2005; and Brette, 2006, for refined versions of such an algorithm.) This decision process clearly comes at a cost and becomes unwieldy for networks of high connectivity: if each neuron is receiving input spikes at a conservative average rate of 1 Hz from each of 10^4 synapses, it needs to process a spike every 0.1 ms, and this limits the characteristic integration step size. Therefore, time-driven algorithms have been found useful for the simulation of large, highly connected networks. Here, each neuron is updated on an equidistant time grid, and the emission and absorption times of spikes are restricted to the grid (see section 3). The temporal spacing of the grid is called computation time step h. Consider a network of 10^5 neurons as described above. At a computation time step of 0.1 ms, a time-driven algorithm carries out 10^9 neuron updates per second, the same number as required for the event-driven algorithm. In this situation, the time-driven scheme is necessarily faster than the event-driven scheme because the costs of the actual updates are the same and there is no overhead caused by the scheduling of events.

However, in this letter, we criticize this view and argue that in order to arrive at a relevant measure of efficiency, simulation time should be analyzed as a function of the integration error rather than the update interval. Whether a time-driven or event-driven scheme yields a better perfomance from this perspective depends on the required accuracy of the simulation and the network spike rate, and is not immediately apparent from considerations of complexity.

In the time-driven framework, Rotter and Diesmann (1999) showed that for a wide class of neuron models, the linearity of the subthreshold dynamics can be exploited to integrate the neuron state exactly from one grid point to the next by performing a single matrix vector multiplication. Here, the computation time step simultaneously determines the accuracy with which incoming spikes influence the subthreshold dynamics and the timescale at which threshold crossings can be detected. However, Hansel, Mato, Meunier, and Neltner (1998) showed that forcing spikes onto the grid can significantly distort the synchronization dynamics of certain networks. Reducing the computation step ameliorates the problem only slowly, as the integration error declines linearly with h (see section 8.4.1). The problem was solved (Hansel et al., 1998; Shelley & Tao, 2001) by interpolating the membrane potential between grid points to give a better approximation of the time of threshold crossing and evaluating the effect of incoming spikes on the neuronal state in continuous time.

In this work, we demonstrate that the techniques developed for the exact integration of the subthreshold dynamics (Rotter & Diesmann, 1999) and for the interpolation of spike times (Hansel et al., 1998; Shelley & Tao, 2001) can be successfully combined. By requiring that the minimal synaptic propagation delay be at least as large as the computation time step, all events can be queued at their target neurons rather than relying on a central event queue to maintain causality in the network. This reduces the complexity of the global scheduling algorithm—that is, deciding which neuron should be updated, and how far—to the simple time-driven case, whereby each neuron in turn is advanced in time by a fixed amount. Therefore, the global overhead costs are no more than in a traditional discrete-time simulation, and yet on the level of the individual neuron, spikes can be processed and emitted in continuous time with the accuracy of an event-driven algorithm. This approach represents a hybridization of traditional time-driven and event-driven algorithms: the scheme is time driven on the global level to advance the system time but event driven on the level of the individual neurons.

The exact integration method is predicated on the linearity of the subthreshold dynamics. We show that this property can be further exploited, as the order of incoming events is not relevant for calculating the neuron state. This completely removes the need for storing and sorting individual events, and therefore also for dynamic data structures, while maintaining the high precision of the event-driven approach.

We illustrate these ideas by comparing three implementations of the same widely used integrate-and-fire neuron model. As in previous work, the scaling behavior of the integration error is considered a function of the computational resolution. However, in contrast to these works, we also analyze the run time and memory consumption of a large neuronal network model as a function of integration error, thus defining a measure of efficiency that can be applied to any competing model implementations. This analysis reveals that the novel scheme of embedding continuous-time implementations in a discrete-time framework can in many cases result in simulations that are both more accurate and faster than a given purely discrete-time simulation. This is possible because the new scheme achieves the same accuracy at larger computation time steps. Depending on the rate of events to be processed by the neuron, the gain in simulation speed due to an increased step size can more than compensate for the increased complexity of processing continuous-time input events. The scheme presented is well suited for distributed computing.

In section 2, we describe the neuron model used as an example in the remainder of the article, and then we review the techniques for integrating the dynamics of a neural network in discrete time steps in section 3. Subsequently, in section 4, we present two implementations solving the single-neuron dynamics between grid points but handling the incoming and emitted spikes in continuous time. The performance of these implementations with respect to integration error, run time, and memory requirements is analyzed in section 5. We show that the choice of which implementation should be used for a given problem depends on a trade-off between these factors. The concepts of time-driven and event-driven simulation of large neural networks are discussed in section 6 in the light of our findings. The numerical techniques underlying the reported results are given in the appendix.

The conceptual and algorithmic work described here is a module in our long-term collaborative project to provide the technology for neural systems simulations (Diesmann & Gewaltig, 2002).

Preliminary results have been presented in abstract form (Morrison, Hake, Straube, Plesser, & Diesmann, 2005).

2 Example Neuron Model

Although the methods in this letter can be applied to any neuron model reducible to a system of linear differential equations, for clarity, we compare various implementations of one particular physical model: a current-based integrate-and-fire neuron with postsynaptic currents represented as α-functions. The dynamics of the membrane potential V is:

$$\dot{V} = -\frac{V}{\tau_m} + \frac{1}{C} I,$$

where τ_m is the membrane time constant, C is the capacitance of the membrane, and I is the input current to the neuron. The current arises as a superposition of the synaptic currents and any external current. The time course of the synaptic current ι due to one incoming spike is

$$\iota(t) = \hat{\iota}\frac{e}{\tau_\alpha} t e^{-t/\tau_\alpha},$$

where $\hat{\iota}$ is the peak value of the current and τ_α is the rise time. When the membrane potential reaches a given threshold value Θ, the membrane potential is clamped to zero for an absolute refractory period τ_r. The values for these parameters used in this article are τ_m, 10 ms; C, 250 pF; Θ, 20 mV; τ_r, 2 ms; $\hat{\iota}$, 103.4 pA; and τ_α, 0.1 ms.

3 Exact Integration of Subthreshold Dynamics in a Discrete Time Simulation

The dynamics of the neuron model described in section 2 is linear and can therefore be reformulated to give a particularly efficient implementation for a discrete-time simulation (Rotter & Diesmann, 1999). We refer to this traditional, discrete-time approach as the grid-constrained implementation. Making the substitutions

$$y^1 = \frac{d}{dt}\iota + \frac{1}{\tau_\alpha}\iota$$
$$y^2 = \iota$$
$$y^3 = V,$$

where y^i is the ith component of the state vector $\mathbf{y}$, we arrive at the following system of linear equations:

$$\dot{\mathbf{y}} = \mathbf{A}\mathbf{y} = \begin{bmatrix} -\frac{1}{\tau_\alpha} & 0 & 0 \\ 1 & -\frac{1}{\tau_\alpha} & 0 \\ 0 & \frac{1}{C} & -\frac{1}{\tau_m} \end{bmatrix} \mathbf{y}, \quad \mathbf{y}(0) = \begin{bmatrix} \hat{\iota}\frac{e}{\tau_\alpha} \\ 0 \\ 0 \end{bmatrix},$$

where $\mathbf{y}(0)$ is the initial condition for a postsynaptic potential originating at time $t = 0$. The exact solution of this system is given by $\mathbf{y}(t) = \mathbf{P}(t)\mathbf{y}(0)$, where $\mathbf{P}(t) = e^{\mathbf{A}t}$ denotes the matrix exponential of $\mathbf{A}t$, which is an exact mathematical expression (see, e.g., Golub & van Loan, 1996). For a fixed time step h, the state of the system can be propagated from one grid position to the next by $\mathbf{y}_{t+h} = \mathbf{P}(h)\mathbf{y}_t$. This is an efficient method because $\mathbf{P}(h)$ is constant and has to be calculated only once at the beginning of a simulation. Moreover, $\mathbf{P}(h)$ can be obtained in closed form (Diesmann, Gewaltig, Rotter,

& Aertsen, 2001), for example, using symbolic algebra software such as Maple (Heck, 2003) or Mathematica (Wolfram, 2003), and can therefore be evaluated by simple expressions in the implementation language (see also section A.3). The complete update for the neuron state may be written as

$$\mathbf{y}_{t+h} = \mathbf{P}(h)\mathbf{y}_t + \mathbf{x}_{t+h}, \tag{3.1}$$

assuming incoming spikes are constrained to the grid, as the linearity of the system permits the initial conditions for all spikes arriving at a given grid point to be lumped together into one term, $\mathbf{x}_{t+h}$

$$\mathbf{x}_{t+h} = \begin{bmatrix} \frac{e}{\tau_\alpha} \\ 0 \\ 0 \end{bmatrix} \sum_{k \in S_{t+h}} \hat{\imath}_k. \tag{3.2}$$

Here S_{t+h} is the set of indices $k \in 1, \ldots, K$ of synapses that deliver a spike to the neuron at time $t + h$, and $\hat{\imath}_k$ represents the "weight" of synapse k. Note that the $\hat{\imath}_k$ may be arbitrarily signed and may also vary over the course of the simulation. The new neuron state $\mathbf{y}_{t+h}$ is the exact solution to the subthreshold neural dynamics at time $t + h$, including all events that arrive at time $t + h$. This assumes that the neuron does not itself produce a spike in the interval $(t, t + h]$.

If the membrane potential y^3_{t+h} exceeds the threshold value Θ, the neuron communicates a spike event to the network with a time stamp of $t + h$; the membrane potential is subsequently clamped to 0 in the interval $[t + h, t + h + \tau_r]$ (see Figure 1A). The earliest grid point at which a neuron could produce its next spike is therefore $t + 2h + \tau_r$. Note that for a grid-constrained implementation, τ_r must be an integer multiple of h, because the membrane potential is evaluated only at grid points, and we define it to be nonzero.

3.1 Computational Requirements. In order to preserve causality, it is necessary that there is a minimal synaptic delay of h. Otherwise, if a neuron spiked at time $t + h$ and its synapses had a propagation delay of 0, then this event would seem to arrive at some of its targets at $t + h$ and at some of them at $t + 2h$, depending on the order in which the neurons are updated. In practice, simulations are generally performed with synaptic delays that are greater than the time step h, and so some technique must be used to store events that have already been produced by a neuron but are not due to arrive at their targets for several time steps. In a grid-constrained simulation, only delays that are an integer multiple of h can be considered because incoming spikes can be handled only at grid points. Consequently, pending events can be stored in a data structure analogous to a looped tape device (see Morrison, Mehring, et al., 2005). If a neuron emits a spike at time t that has a

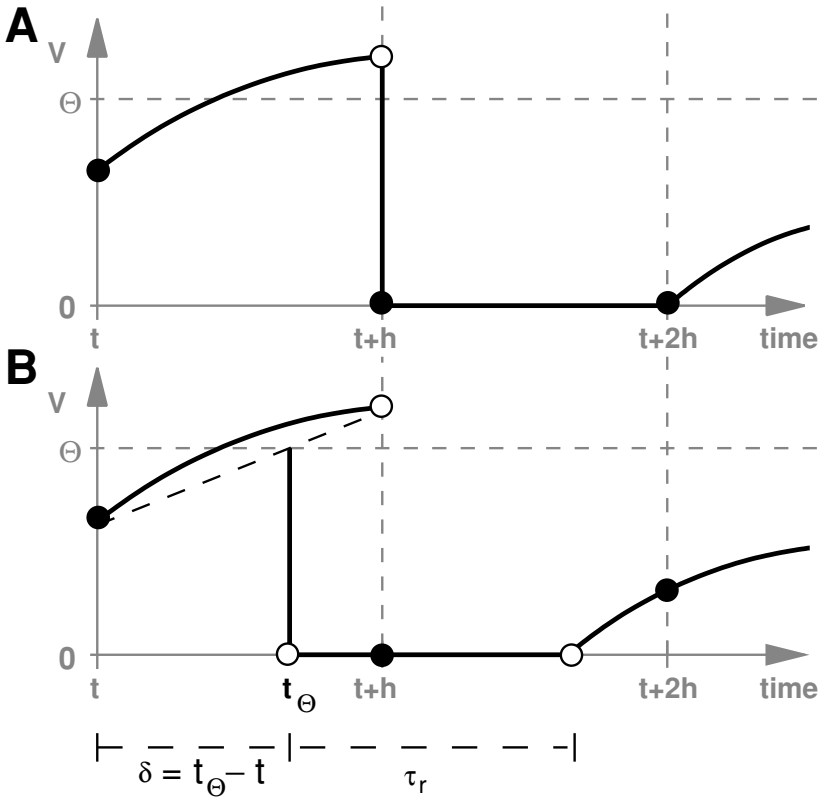

Figure 1: Spike generation and refractory periods for a grid-constrained (A) and a continuous-time implementation (B). The spike threshold Θ is indicated by the dashed horizontal line. The solid black curve shows the membrane potential time course for a neuron subject to a suprathreshold constant input current leading to a threshold crossing in the interval $(t, t+h]$. The gray vertical lines indicate the discrete time grid with spacing h. The refractory period τ_r in this example is set to its minimal value h. Filled circles denote observable values of the membrane potential; unfilled circles denote supporting points that are not observable. (A) In the grid-constrained implementation, the spike is emitted at $t+h$. During the refractory period τ_r, the membrane potential is clamped to zero. (B) In the continuous-time implementation, the threshold crossing is found by interpolation (here linear: black dashed line) at time t_Θ. The spike is emitted with the time stamp $t+h$ and with an offset with respect to t of $\delta = t_\Theta - t$. The neuron is refractory from t_Θ until $t_\Theta + \tau_r$, during which period the membrane potential is clamped to zero. At grid point $t+2h$, a finite membrane potential can be observed.

delay of d, the simulation algorithm waits until all neurons have completed their updates for the integration step $(t-h, t]$ and then delivers the event to its target(s). The event is placed in the tape device of the target neuron d/h segments on from the current reading position. It will then become visible to the target neuron at the grid point $t+d$, when the neuron is performing the update for the integration step $(t+d-h, t+d]$. Recalling that the initial conditions for all events arriving at a given time may be lumped together (see equation 3.2) and that two of the three components of the initial conditions vector are zero, the segments of the looped taped device can be very simple. Each segment contains just one value, which is incremented by the weight of every spike event delivered there. In other words, when the neuron performs the integration step $(t, t+h]$, the segment visible to the reading head contains the first component of $\mathbf{x}_{t+h}$ up to the scaling factor of $\frac{e}{\tau_\alpha}$.

In terms of memory, the looped tape device needs as many segments as computation steps are required to cover the maximum synaptic delay, plus an additional segment to represent the events arriving at $t+h$. Therefore, performing a given simulation with a smaller time step will require more memory. The model described in section 2 has only a single synaptic dynamics and so requires only one tape device; models using multiple types of synaptic dynamics can be implemented in this framework by providing them with the corresponding number of tape devices.

4 Continuous-Time Implementation of Exact Integration

4.1 Canonical Implementation. The most obvious way to reconcile exact integration and precise spike timing within a discrete-time simulation is to store the precise times of incoming events. In order to represent this information, an offset must be assigned to each spike event in addition to the time stamp. This offset is measured from the beginning of the interval in which the spike was produced: a spike generated at time $t+\delta$ receives a time stamp of $t+h$ and an offset of δ (see Figure 1B). Given a sorted list of event offsets $\{\delta_1, \delta_2, \cdots, \delta_n\}$ with $\delta_i \leq h$, which become visible to a neuron in the step $(t, t+h]$, exact integration of the subthreshold dynamics can be performed from the beginning of the time step to the beginning of the list:

$$\mathbf{y}_{t+\delta_1} = \mathbf{P}(\delta_1)\mathbf{y}_t + \mathbf{x}_{\delta_1};$$

then along the list:

$$\mathbf{y}_{t+\delta_2} = \mathbf{P}(\delta_2 - \delta_1)\mathbf{y}_{t+\delta_1} + \mathbf{x}_{\delta_2}$$

$$\vdots$$

$$\mathbf{y}_{t+\delta_n} = \mathbf{P}(\delta_n - \delta_{n-1})\mathbf{y}_{t+\delta_{n-1}} + \mathbf{x}_{\delta_n};$$

and finally from the end of the list to the end of the time step:

$$\mathbf{y}_{t+h} = \mathbf{P}(h - \delta_n)\mathbf{y}_{t+\delta_n}.$$

The final term $\mathbf{y}_{t+h}$ is the exact solution for the neuron dynamics at time $t + h$. This sequence is illustrated in Figure 2B. This is assuming that the neuron does not produce a spike or emerge from its refractory period during this interval. These special cases are described in more detail below.

4.1.1 Spike Generation. In the grid-constrained implementation, the neuron state is inspected at the end of each time step to see if it meets its spiking criteria. In the case of the neuron model described in section 3, the criterion is $y^3 \geq \Theta$, where Θ is the threshold. In this implementation, the neuron state can be inspected after every step of the process described in section 4.1. If $y^3_{t+\delta_i} < \Theta$ and $y^3_{t+\delta_{i+1}} \geq \Theta$, then the membrane potential of the neuron reached threshold between $t + \delta_i$ and $t + \delta_{i+1}$. As the dynamics of this model is not invertible, the time t_Θ of this threshold passing can be determined only by interpolating the membrane potential in the interval $(t + \delta_i, t + \delta_{i+1}]$. For this article, linear, quadratic, and cubic interpolation schemes were investigated.

After the threshold crossing, the neuron is refractory for the duration of its refractory period τ_r. The membrane potential y^3 is set to zero and need not be calculated during this period, although the other components of the neuron state continue to be updated as in section 4.1.

At the end of the time interval, an event is dispatched with a discrete-time stamp of $t + h$ and an offset of $\delta = t_\Theta - t$ (see Figure 1B).

4.1.2 Emerging from the Refractory Period. The neuron emerges from its refractory period in the time step defined by $t < t_\Theta + \tau_r \leq t + h$ (see Figure 1B). In contrast to the grid-constrained implementation, τ_r does not have to be an integer multiple of h. For a continuous time τ_r, a grid position t can always be found such that $t_\Theta + \tau_r$ comes within $(t, t + h]$. However, the implementation is simpler when τ_r is a nonzero integer multiple of h. To calculate the neuron state at time $t + h$ exactly, the interval is divided into two subintervals: $(t, t_\Theta + \tau_r]$ and $(t_\Theta + \tau_r, t + h]$. In the first period, the neuron is still refractory, so when performing the exact integration along the incoming events as in section 4.1, y^3 is not calculated and remains at zero. At the end of this period, the neuron state $\mathbf{y}_{t_\Theta+\tau_r}$ is the exact solution for the dynamics at the end of the refractory period. In the second period, the neuron is no longer refractory, and so the exact integration can be performed as usual (including the calculation of y^3). The neuron state $\mathbf{y}_{t+h}$ is therefore an exact solution to the neuron dynamics at time $t + h$.

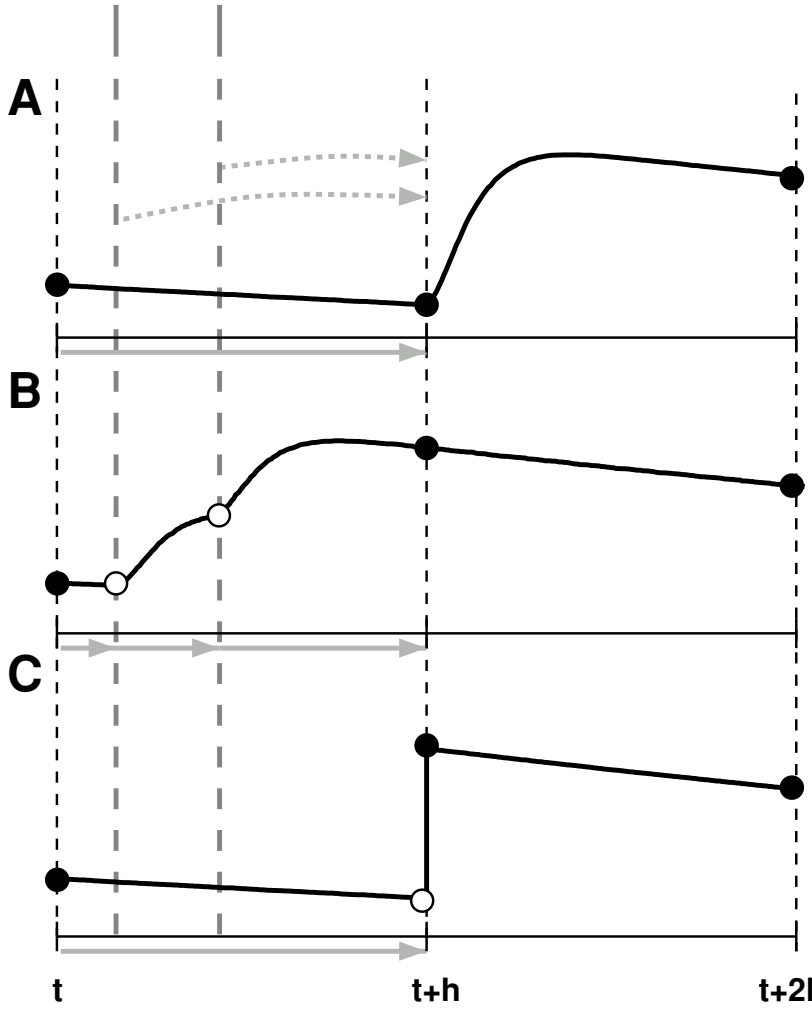

Figure 2: Handling of incoming spikes in the grid-constrained (A), the canonical (B), and the prescient (C) implementations. In each panel, the solid curve represents the excursion of the membrane potential in response to two incoming spikes (gray vertical lines). Filled circles denote observable values of the membrane potential; unfilled circles denote supporting points that are not observable. The gray horizontal arrows beneath each panel indicate the propagation steps performed during the time step $(t, t + h]$. Dashed arrows in A indicate that in the grid-constrained implementation, input spikes are effectively shifted to the next point on the grid. The observable membrane potentials following spike impact are identical and exact in B and C but differ from those in A.

4.1.3 Computational Requirements. As in the grid-constrained implementation, a minimum spike propagation delay of h is necessary in order to ensure that all events due at the neuron between t and $t + h$ have arrived by the time the neuron performs its update for that interval. The simple looped event buffer described in section 3.1 must be extended to store the weights and offsets for incoming events separately. As the number of events arriving in an interval cannot be known ahead of time, this structure must be capable of dynamic memory allocation, which reduces its cache effectiveness. However, information about the minimum propagation delay in the network can be utilized to streamline the data structure so that its size does not depend on h, which compensates to some extent for the use of dynamic memory. Finally, as the events may arrive in any order, the buffer must also be capable of sorting the events with respect to increasing offset, which for a general-purpose sorting algorithm has complexity $\mathcal{O}(n \log n)$. An alternative representation of the timing data is a priority queue; in practice, this was not quite as efficient as the looped tape device. In contrast to the grid-constrained implementation, delays can now be represented in continuous time. A spike arrival time $t + \delta + d$, where $t + \delta$ is a continuous spike generation time and d is a continuous time delay, can always be decomposed into a discrete grid point $t + d'$ and a continuous offset δ'. However, for notational and implementational convenience (see section 6), we assume d to be a nonzero integer multiple of h.

4.2 Prescient Implementation. In the implementation described in section 4.1, the neuron state at the end of a time step is calculated by integrating along a sorted list of events. However, as the subthreshold dynamics is linear, it is not dependent on the order of events. In this section, an implementation is presented that exploits this fact to reduce the computational complexity and dynamic memory requirements of the canonical implementation.

4.2.1 Receiving an Event. Consider a spike event generated during the step $(t - h, t]$ with offset δ and transmitted with delay d. This spike will be processed during the update step $(t + d - h, t + d]$, its effect being observable for the first time at $t + d$. Since the correct spike arrival time is $t + d - h + \delta$, when the algorithm delivers the spike to the neuron we evolve the effect of the spike input from the arrival time to the end of the interval at $t + d$ using

$$\tilde{\mathbf{y}}_{t+d} = \mathbf{P}(h - \delta)\mathbf{x}.$$

Therefore, instead of storing the entire event as for the canonical implementation, the three components of its effect on the neuron state can be stored in an event buffer at the position d instead. Due to the linearity of the system,

these components can be summed for all events due to arrive at the neuron in a given time step, regardless of the order in which the algorithm delivers them to the neuron or the order in which they are to become visible to the neuron. As we exploit the fact that the effect of an event on the neuron can be calculated before the event becomes visible to the neuron, we call this the prescient implementation.

4.2.2 Calculating the Subthreshold Dynamics. At the beginning of each time interval $(t, t + h]$, the total effect of all events arriving within that step on the three components of the neuron state at time $t + h$ is already stored in the event buffer. Calculating the neuron state at the end of the time step is therefore simple:

$$\mathbf{y}_{t+h} = \mathbf{P}(h)\mathbf{y}_t + \tilde{\mathbf{y}}_{t+h}.$$

The new neuron state $\mathbf{y}_{t+h}$ is the exact solution to the neuron dynamics at time $t + h$, including all events that arrived within the step $(t, t + h]$. This is depicted in Figure 2C. As with the canonical implementation, there are two special cases that need to be treated with more care: a time step in which a spike is generated and one in which the neuron emerges from its refractory period.

4.2.3 Spike Generation. The process that generates a spike is very similar to that for the canonical implementation described in section 4.1.1. In this case, as the timing of the incoming events is no longer known, the neuron state can be inspected only at the end of the time step rather than at each incoming event, and so the length of the interpolation interval is h rather than the interspike interval of incoming events.

4.2.4 Emerging from the Refractory Period. As for the canonical implementation, the time step in which the neuron emerges from its refractory period is divided into two subintervals: $(t, t_\Theta + \tau_r]$ and $(t_\Theta + \tau_r, t + h]$. Setting $t_{em} = t_\Theta + \tau_r - t$, the neuron state at the end of the refractory period can be calculated as follows:

$$\begin{aligned}\mathbf{y}_{t+t_{em}} &= \mathbf{P}(t_{em})\mathbf{y}_t \\ y^3_{t+t_{em}} &\leftarrow 0.\end{aligned}$$

Having emerged from the refractory period, the membrane potential is no longer clamped to zero and can develop normally during the remainder of the time step (see Figure 1B):

$$\mathbf{y}_{t+h} = \mathbf{P}(h - t_{em})\mathbf{y}_{t+t_{em}} + \tilde{\mathbf{y}}_{t+h}.$$

However, this overestimates the effect of the events arriving in the time interval $(t, t+h]$ on the membrane potential. The summation of the components of these events was predicated on the assumption of linear dynamics, but as the membrane potential is clamped to zero until $t + t_{\text{em}}$, this assumption does not hold. Any events arriving at the neuron before its emergence from its refractory period should have no effect on its membrane potential before this point, yet adding the full value of the third component of $\tilde{\mathbf{y}}_{t+h}$ assumes that they do. As a corrective measure, the effect of the new events on the membrane potential can be considered to be linear within the small interval $(t, t+h]$, and the membrane potential can be adjusted accordingly:

$$y^3_{t+h} \leftarrow y^3_{t+h} - \gamma \tilde{y}^3_{t+h},$$

with $\gamma = t_{\text{em}}/h$.

4.2.5 Computational Requirements. As in the grid-constrained and canonical implementations, a minimum spike propagation delay of h is required to preserve causality. The looped-tape device described in section 3.1 needs to be able to store the three components of the neuron state rather than just the weight of the incoming events. Alternatively, three event buffers can be used, capable of storing one component each. Unlike the buffer devices for the canonical implementation, they need to store only one value per time step rather than one for each incoming spike, so there is no time overhead for sorting the values. Moreover, they do not require dynamic memory allocation and so are more cache effective.

5 Performance

In order to compare error scaling for the different implementations and interpolation orders, a simple single-neuron simulation was chosen. As the system is deterministic and nonchaotic, reducing the computation time step h causes the simulation results to converge to the exact solution, so error measures can be well defined. To investigate the costs incurred by simulating at finer resolutions or using computationally more expensive off-grid neuron implementations, a network simulation was chosen. This is fairer than a single-neuron simulation, as the run-time penalties of applications requiring more memory will come into play only if the application is large enough not to fit easily into the processor's cache memory. Furthermore, it is only when performing network simulations that the bite of long simulation times per neuron is really felt.

5.1 Single-Neuron Simulations. Each experiment consisted of 40 trials of 500 ms each, during which a neuron of the type described in section 2 was stimulated with a constant excitatory current of 412 pA and unique

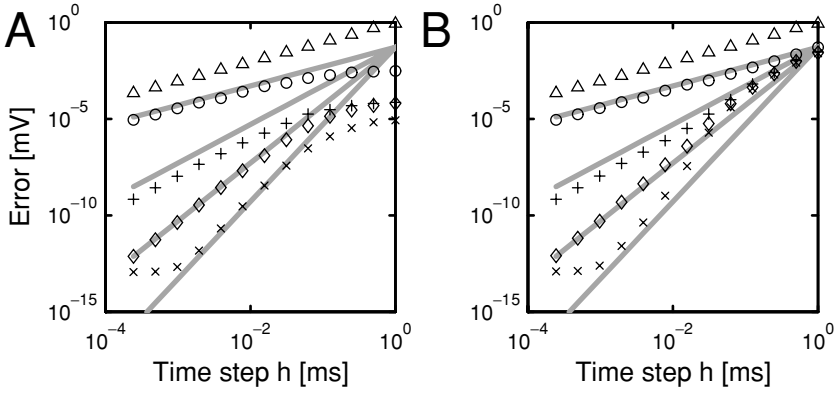

Figure 3: Scaling of error in membrane potential as a function of the computational resolution in double logarithmic representation. (A) Canonical implementation. (B) Prescient implementation. No interpolation, circles; linear interpolation, plus signs; quadratic interpolation, diamonds; cubic interpolation, multiplication signs. In both cases, the triangles show the behavior of the grid-constrained implementation, and the gray lines indicate the slopes expected for scaling of orders first to fourth with an arbitrary intercept of the vertical axis.

realizations of an excitatory Poissonian spike train of 1.3×10^4 Hz and an inhibitory Poissonian spike train of 3×10^3 Hz. The spike times of the Poissonian input trains were represented in continuous time. Parameters are as in section 2, but the peak value of the current resulting from an inhibitory spike was a factor of 6.25 greater than that of an excitatory spike to ensure a balance between excitation and inhibition. The output firing rate was 12.7 Hz. The experiment was repeated for each implementation with each interpolation order over a wide range of computational resolutions. As the membrane potential and spike times cannot be calculated analytically for this protocol, the canonical implementation with cubic interpolation at the finest resolution (2^{-13} ms ≈ 0.12 μs) was defined to be the reference simulation for each realization of the input spike train. As a measure of the error in calculating the membrane potential, the deviation of the actual membrane potential from the reference membrane potential was sampled every millisecond for all the trials.

In Figure 3, the median of these deviations is plotted as a function of the computational resolution in double logarithmic representation. In both the canonical implementation (see Figure 3A) and the prescient implementation (see Figure 3B), the same scaling behavior can be seen: for an interpolation order of n, the error in membrane potential scales with order $n + 1$ (see section A.4). The error has a lower bound at 10^{-14}, which can be seen for very fine resolutions using cubic interpolation. This represents the greatest

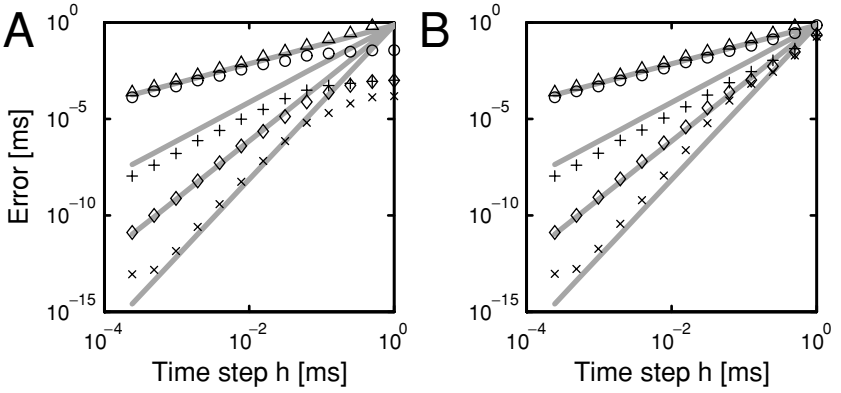

Figure 4: Scaling of error in spike times as a function of the computational resolution. (A) Canonical implementation. (B) Prescient implementation. Symbols and lines as in Figure 3.

numerical precision possible for this physical quantity using the standard representation of floating-point numbers (see section A.1). Interestingly, the error for the canonical implementation also saturates at coarse resolutions. This is because interpolation is performed between incoming events rather than across the whole time step, as in the case of the prescient implementation. Consequently, the effective computational resolution cannot be any coarser than the average interspike interval of the incoming spike train (in this case, 1/16 ms), and this determines the maximum error of the canonical implementation. Note that the error of the grid-constrained implementation scales in the same way as that of the canonical and prescient implementations with no interpolation. However, due to the fact that incoming spikes are forced to the grid, the absolute error is greater for this implementation.

The accuracy of a simulation is not determined by the membrane potential alone; the precision of spike times is of at least as much relevance. The median of the differences between the actual and the reference spike times is shown in Figure 4. As with the error in membrane potential, using an interpolation order of n results in a scaling of order $n+1$, and the error has a lower bound that is exhibited at very fine resolution when using cubic interpolation. Furthermore, a similar upper bound on the error is observed for the canonical implementation at coarse resolutions. However, in this case, the grid-constrained implementation exhibits not only the same scaling, but a similar absolute error to the continuous-time implementations with no interpolation. Recalling that all the simulations receive identical continuous-time Poisson spike trains, the only difference remaining between the grid-constrained implementation and the continuous-time implementations without interpolation is that the former ignores the offset of the incoming spikes and treats them as if they were produced on the

grid, whereas the latter process the incoming spikes precisely. This reveals that handling incoming spikes precisely confers no real advantage if outgoing spikes are generated without interpolation and thereby forced to the grid. One might therefore conclude that the precise handling of incoming spikes is unnecessary and that the single-neuron integration error could be significantly improved just by performing an appropriate interpolation to generate spikes, while treating incoming spikes as if they were produced on the grid. In fact, this is not the case. If incoming spikes are treated as if on the grid, the error in the membrane potential decreases only linearly with h, thus limiting the accuracy of higher-order methods in determining threshold crossings. This is corroborated by simulations (data not shown). Substantial improvement in the accuracy of single-neuron simulations requires both techniques: precise handling of incoming spikes and interpolation of outgoing spikes.

5.2 Network Simulations. In order to determine the efficiency of the various implementations, a balanced recurrent network was adapted from Brunel (2000). The network contained 10,240 excitatory and 2560 inhibitory neurons and had a connection probability of 0.1, resulting in a total of 15.6×10^6 synapses. The inhibitory synapses were a factor of 6.25 stronger than the excitatory synapses, and each neuron received a constant excitatory current of 412 pA as its sole external input. Membrane potentials were initialized to values chosen from a uniform random distribution over $[-\Theta/2, 0.99\Theta]$. In this configuration, the network fires with approximately 12.7 Hz in the asynchronous irregular regime, which recreates the input statistics used in the single-neuron simulations. The synaptic delay was 1 ms, and the network was simulated for 1 biological second.

The simulation time and memory requirements for the network simulation described above are shown in Figure 5. For the simulation time (see Figure 5A), it can be seen that at coarse resolutions, the grid-constrained implementation is significantly faster than the prescient implementation, which in turn is faster than the canonical implementation. This is due to the fact that the cost of processing spikes is essentially independent of the computational resolution and manifests as an implementation-dependent constant contribution to the simulation time, which is particularly dominant at coarse resolutions. The difference in speed between the canonical and prescient implementations results from the use of dynamic as opposed to static data structures, and to a lesser extent from the cost of sorting incoming spikes in the canonical implementation. As the computation time step decreases, the simulation times converge, because the cost of updating the neuron dynamics in the absence of events, which is the same for all implementations, is inversely proportional to the resolution and so manifests as a scaling with exponent -1 at small computation time steps (see Figure 5A). It is clear that in general, at the same computation time step, the continuous-time implementations must be slower than the grid-constrained

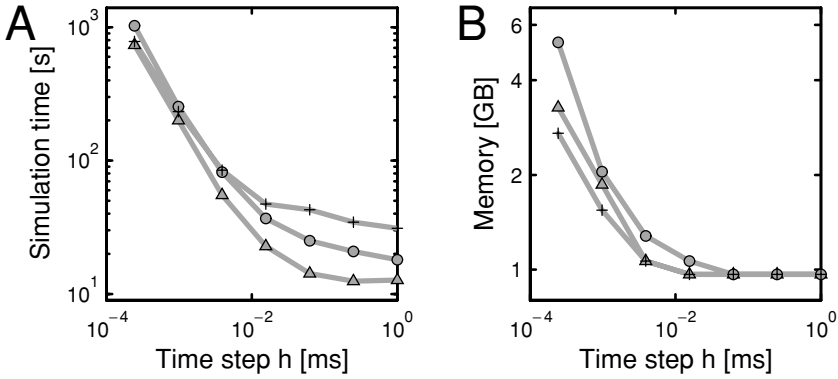

Figure 5: Simulation time (A) and memory requirements (B) for a network simulation as functions of computational resolution in double logarithmic representation. Triangles, grid-constrained neuron; plus signs, canonical implementation with cubic interpolation; circles, prescient implementation with cubic interpolation. Other interpolation orders for the canonical and the prescient implementations result in practically identical curves and are therefore not shown. For details of the simulation, see the text.

implementation, as the former perform a propagation for every incoming spike and the latter does not. The increased costs concomitant with higher interpolation orders proved to be negligible in a network simulation.

An increase in memory requirements can be observed for all implementations (see Figure 5B) as the resolutions become finer. Although all implementations require much the same amount of memory at coarser resolutions, for finer resolutions, the canonical implementation requires the least memory, followed by the grid-constrained implementation, and the prescient implementation requires the most. It is clear that for a wide range of resolutions, the memory required by the rest of the network, specifically the synapses, dominates the total memory requirements (Morrison, Mehring, et al., 2005). As the resolution becomes finer, the memory required for the input buffers for the neurons plays a greater role. The spike buffer for the canonical implementation is independent of the resolution (see section 4.1.3), and so it might seem that it should not require more memory at finer resolution. However, all implementations tested here also have a buffer for a piece-wise constant input current. In addition to this, the grid-constrained implementation has one buffer for the weights of incoming spikes, and the prescient implementation has three buffers—one for each component of the state vector. All of these buffers require memory in inverse proportion to the resolution, thus explaining the ordering of the curves. Generally a smaller application is more cache effective than a larger one, and this may explain why the canonical implementation exhibits slightly lower

simulation times than the other implementations at very fine resolutions (see Figure 5A).

5.3 Conjunction of Integration Error and Run-Time Costs. The considerations in section 5.2 of how the simulation time and memory requirements increase with finer resolutions are of limited practical relevance to a scientist with a particular problem to investigate. More interesting in this case is how much precision bang you get for your simulation time buck. Unlike the single neuron, however, the network described above is a chaotic system (Brunel, 2000). Any deviation at all between simulations will lead, in a short time, to totally different results on the microscopic level, such as the evoked spike patterns. Such a deviation can even be caused by differences in the tiny round-off errors that occur if floating-point numbers are summed in a different order. Because of this, these simulations do not converge on the microscopic level as the single-neuron simulations do, and for that reason the so-called accuracy of a simulation cannot be taken at face value.

We therefore relate the cost of network simulations to the accuracy of single-neuron simulations with comparable input statistics. In Figure 6, the simulation time and memory requirements data from Figure 5 are combined with the accuracy of the corresponding single-neuron simulations shown in Figure 4, thus eliminating the computational resolution as a parameter. Figure 6A shows the simulation time as a function of spike time error for the three implementations. This graph can be read in two directions: horizontally and vertically. By reading the graph horizontally, we can determine which implementation will give the best accuracy for a given affordable simulation time. Reading the graph vertically allows us to determine which implementation will result in the shortest simulation time for a given acceptable accuracy. Concentrating on the latter interpretation, it can be seen from the intersection of the lines corresponding to the prescient- and grid-based implementations (vertical dashed line in Figure 5A) that if an error greater than 2.3×10^{-2} ms is acceptable, the grid-constrained implementation is faster. For better accuracy than this, the prescient implementation is more effective. If an appropriate time step is chosen, the prescient implementation can simulate more accurately and more quickly than a given grid-constrained simulation in this regime. Only for very high accuracy can a lower simulation time be achieved using the canonical implementation.

Similarly, Figure 6B, which shows the memory requirements as a function of spike-time error in double logarithmic representation, can be read in both directions. This shows qualitatively the same relationship as in Figure 6A, but the point at which one would switch from the prescient implementation to the canonical implementation in order to conserve memory occurs for larger errors. The flatness of the curves for the continuous-time implementations shows that it is possible to increase the accuracy of a simulation considerably without having to worry about the memory requirements.

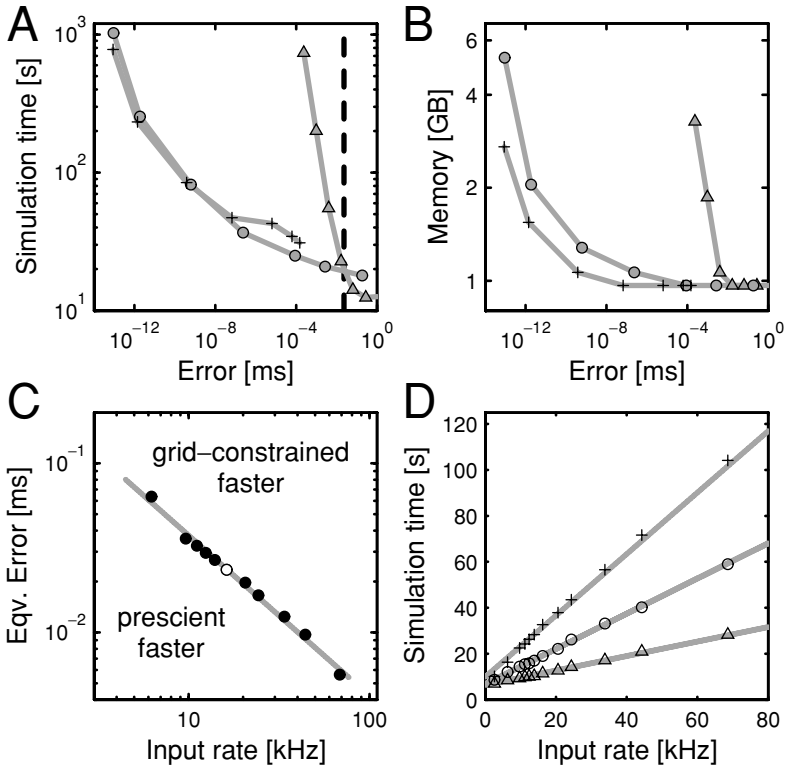

Figure 6: Analysis of simulation time and memory requirements for a network simulation as functions of spike-time error for the single-neuron simulation and input spike rate. (A) Simulation time as a function of spike-time error in double logarithmic representation. Triangles, grid-constrained neuron; plus signs, canonical implementation with cubic interpolation; circles, prescient implementation with cubic interpolation. Data combined from Figure 4 and Figure 5. For errors smaller than 2.3×10^{-2} ms (vertical dashed line), a continuous-time implementation with an appropriately chosen computation step size gives better performance. (B) Memory consumption as a function of spike-time error in double logarithmic representation. Symbols as in *A*. (C) Error in spike time for which the prescient and grid-constrained implementations require the same simulation time (for different appropriate choices of h) as a function of input spike rate. The unfilled circle indicates the equivalence error for a network rate of 12.7 Hz (input rate 16 kHz), that is, the intersection marked by the vertical dashed line in *A*. The gray line is a linear fit to the data (slope -0.95). (D) Simulation time as a function of input spike rate for $h = 0.125$ ms, symbols as in *A*. The gray lines are linear fits to the data (slopes 1.3, 0.8 and 0.3 s per kHz from top to bottom).

In general, the point at which the prescient implementation at an appropriate computational resolution can produce a faster and more accurate simulation than a grid-constrained simulation will depend on the rate of events a neuron has to handle. To investigate this relationship, the network described in section 5.2 was simulated with different input currents and inhibitory synapse strengths to generate a range of different firing rates in the asynchronous irregular regime. The single-neuron integration error at which the prescient- and the grid-constrained implementations require the same simulation time is shown in Figure 6C as a function of the input spike rate. This equivalence error depends linearly on the input spike rate, demonstrating that in the parameter space of useful accuracies and realistic input rates, there is a wide regime where the prescient is faster than the grid-constrained implementation. Underlying the benign nature of the comparative effectiveness analyzed in Figure 6C is the dependence of simulation time on the rate of events. For all implementations, the simulation time increases practically linearly with the input spike rate (see Figure 6D), albeit with different slopes.

5.4 Artificial Synchrony. The previous section has shown that in many situations, continuous-time implementations achieve a desired single-neuron integration error more effectively than the grid-constrained implementation. However, continuous-time implementations have an advantage compared to the grid-constrained implementation beyond the single-neuron integration error. In a network simulation carried out with the grid-constrained implementation, the spikes of all neurons are aligned to the temporal grid defined by the computation time step. This causes artificial synchronization between neurons that may distort measures of synchronization and correlation on the network level. To demonstrate this effect, Hansel et al. (1998) investigated a small network of N integrate-and-fire neurons with excitatory all-to-all coupling. Here, we extend their analysis to the three implementations under study and provide a comparison of single-neuron and network-level integration error. In contrast to their study, our network is constructed from the model neuron introduced in section 2. The time constant of the synaptic current τ_α is adjusted to the rise time of the synaptic current in the original model, which was described by a β-function. The synaptic delay d and the absolute refractory period τ_r are set to the maximum computation time step h investigated in this section. Note that this choice of d and τ_r means that these parameters can be represented at all computational resolutions, thus ensuring that all simulations using the grid-constrained implementation are solving the same dynamical system.

Figure 7A illustrates the synchrony in the network as a function of synaptic strength. Following Hansel et al. (1998), synchrony is defined as the variance of the population-averaged membrane potential normalized by the population-averaged variance of the membrane

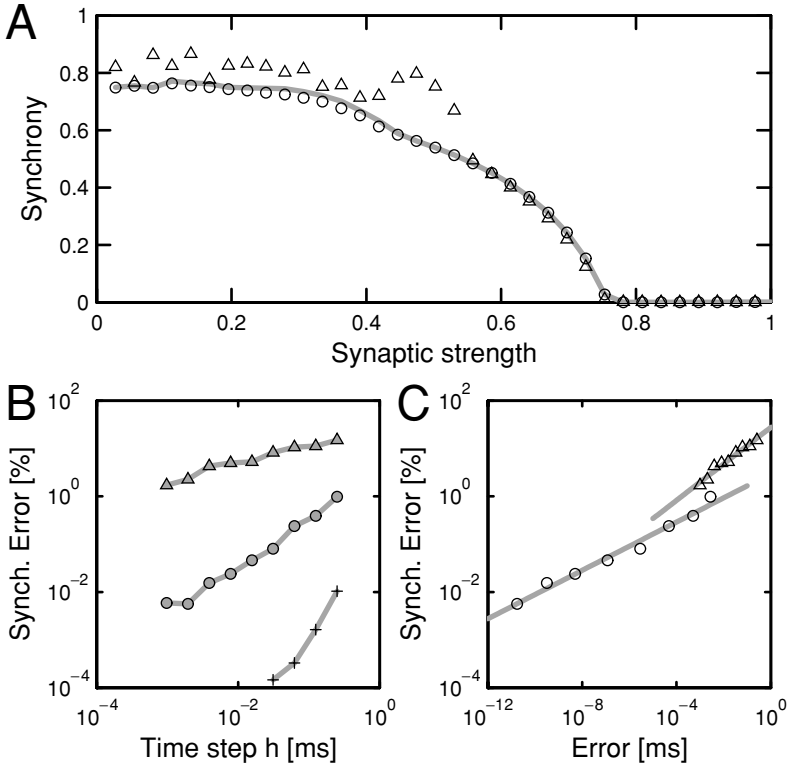

Figure 7: Synchronization error in a network simulation. (A) Network synchrony (see equation 3.3) as a function of synaptic strength in a fully connected network of $N = 128$ neurons ($\tau_\alpha = (3/2)\ln 3$ ms, synaptic delay $d = \tau_r = 0.25$ ms) with excitatory coupling (cf. Hansel et al., 1998). Neurons are driven by a suprathreshold DC $I_0 = 575$ pA, no further external input. The initial membrane potential is $V_i(0) = \frac{\tau_m}{C} I_0[1 - \exp(-\gamma \frac{i-1}{N} \frac{T}{\tau_m})]$, where $i \in 1, \ldots, N$ is the neuron index, T the period in the absence of coupling, and $\gamma = 0.5$ controls the initial coherence. The simulation time is 10 s, and V is recorded in intervals of 1 ms between 5 s and 10 s. Synaptic strength is expressed as the amplitude of a postsynaptic current relative to the rheobase current $I_\infty = (C/\tau_m)\theta = 500$ pA and multiplied by the number of neurons N. Other parameters as in section 2. Canonical implementation as reference ($h = 2^{-10}$ ms, gray curve) and prescient implementation ($h = 2^{-2}$ ms, circles), both with cubic interpolation; grid-constrained implementation ($h = 2^{-5}$ ms, triangles). (B) Synchronization error as a function of the computation time step in double logarithmic representation: grid-constrained implementation, triangles; prescient implementation, circles; canonical implementation, plus signs. (C) Synchronization error as a function of the single-neuron integration error for the grid-constrained and the prescient implementation, same representation as in B. The gray lines are linear fits to the data.

potential time course,

$$S = \left[\left\langle \left\langle V_i(t) \right\rangle_i^2 \right\rangle_t - \left\langle \left\langle V_i(t) \right\rangle_i \right\rangle_t^2\right] \Big/ \left[\left\langle \left\langle V_i(t) \right\rangle_t^2 \right\rangle_i - \left\langle \left\langle V_i(t) \right\rangle_t \right\rangle_i^2\right], \tag{5.1}$$

where $\langle\cdot\rangle_i$ indicates averaging over the N neurons and $\langle\cdot\rangle_t$ indicates averaging over time. This is a measure of coherence in the limit of large N with $S = 1$ for full synchronization and $S = 0$ for the asynchronous state. The grid-constrained implementation exhibits a considerable error in synchrony, which vanishes in approaching the asynchronous regime. The prescient implementation accurately preserves network synchrony even with a significantly larger computation time step.

The error in synchrony is quantified in Figure 7B as the root mean square of the relative deviation of S with respect to the reference solution, estimated over the range of synaptic strength investigated. Note that this includes the asynchronous regime where errors are small in general. The prescient implementation is easily an order of magnitude more accurate than the grid-constrained implementation and is itself outperformed by the canonical implementation. In addition, for the continuous-time implementations, the error in synchrony drops more rapidly with decreasing computational time step h than for the grid-constrained implementation. However, at the same h, different integration methods exhibit a different integration error for the single-neuron dynamics (see Figure 4). Therefore, to accentuate network effects, continuous-time and grid-constrained implementations should be compared at the same single-neuron integration error. To this end, we proceed as follows: the network spike rate is approximately 80 Hz, corresponding to an input spike rate of some 10 kHz. In Figure 7C, the error in network synchrony at a given computational time step h is plotted as a function of the spike time error of a single neuron driven with an input spike rate of approximately 10 kHz, simulated at the same h. For single-neuron errors of 10^{-2} ms and above, the grid-constrained implementation results in considerable errors in network synchrony. Spike timing errors of 10^{-2} ms and below are required for the grid-constrained and the prescient implementations to achieve a synchronization error in the 1% range or better. Interestingly, the grid-constrained implementation exhibits a larger synchronization error than the prescient implementation for identical single-neuron integration errors.

6 Discussion

We have shown that exact integration techniques are compatible with continuous-time handling of spike events within a discrete-time simulation. This combination of techniques achieves arbitrarily high accuracy (up to machine precision) without incurring any extra management costs in the global algorithm, such as a central event queue or looking ahead to see

which neuron will fire next. This is particularly important for the study of large networks with frequent events, as the cost of managing events can become prohibitive (Mattia & Del Giudice, 2000; Reutimann, Giugliano, & Fusi, 2003). We introduced a canonical implementation that illustrates the principles of combining these techniques and a prescient implementation that further exploits the linearity of the subthreshold dynamics. The latter implementation simplifies the neuron update algorithm and requires only static data structures and no queuing, leading to a better time and accuracy performance than the canonical implementation. We compared interpolating polynomials of orders 1 to 3 and discovered that the increased numerical complexity of the higher-order interpolations was not reflected in the run time, which is dominated by other factors. Furthermore, it was shown that the highest-order interpolation performed stably. This suggests that the highest-order interpolation should be used, as the greater accuracy is obtained at negligible cost. We have investigated the nature of the trade-off between accuracy and simulation time/memory and demonstrated that for a large range of input spike rates, it is possible to find a combination of continuous-time implementation and computation time step that fulfills a given maximum error requirement both more accurately and faster than a grid-constrained simulation. This measure of efficiency is based on truly large-scale networks (12,800 neurons, 15.6 million synapses).

The techniques described here have several possible extensions. First, the canonical implementation places no constraints on the neuron model used beyond the physically plausible requirement that the membrane potential is thrice continuously differentiable. It may therefore be used to implement essentially any kind of neuronal dynamics, including neurons with conductance-based synapses. The prescient implementation further requires that the neuron's dynamics is linear; it may thus be used for a wide range of model neurons with current-based synapses. The neuron model we implemented does not have invertible dynamics, and so the determination of its spike time necessarily involves approximation. For some neuron models, it is possible to determine the precise spike time without recourse to approximation, such as the Lapicque model (Tuckwell, 1988) and its descendants (Mirollo & Strogatz, 1990). Such models can, of course, also be implemented in this framework, but they would have only the same precision as a classical event-based simulation if they were canonically implemented (obviously without interpolation); a prescient implementation would be able to represent the subthreshold dynamics exactly on the grid but would entail the use of approximative methods to determine the spike-times. Although we investigated polynomial interpolation, other methods of spike time determination such as Newton's method can be implemented with no change to the conceptual framework.

Second, most constraints imposed in terms of the computational time step h may be relaxed. As indicated in section 4.1.3, the restriction of

delays to nonzero integer multiples of h can be relaxed to any floating-point number $\geq h$. When a neuron spikes, the offset of the spike could then be combined on the fly with the delay to create an integer component and a revised offset, thus allowing the spike to be delivered correctly by the global discrete-time algorithm, and processed correctly by the continuous-time neuron model. This relaxation would come at the memory cost of having to store delays as floating-point numbers rather than integers and the computational cost of having to perform the on-the-fly delay decomposition. Furthermore, h is currently a parameter of the entire network, so it is the same for all neurons in a given simulation. Given that the minimum propagation delay already defines natural synchronization points to preserve causality in the simulation, it would be possible to allow h to be chosen individually for each neuron in the network, or even to use variable time steps, while still maintaining consistency with the global discrete-time algorithm.

Finally, the techniques are compatible with the distributed computing techniques described in Morrison, Mehring, et al. (2005), requiring only that the spike-time offsets are communicated in addition to the indices of the spiking neurons. This increases the bulk but not the frequency of communication, as it is still sufficient to communicate in intervals determined by the minimum propagation delay. A similar minimum delay principle is used by Lytton and Hines (2005), again suggesting a convergence of time-driven and event-driven approaches.

When investigating a particular system, it is worthwhile considering what accuracy is necessary. For the networks described in section 5.2, it would be pointless to simulate with a very small time step, as they are chaotic. As long as the relevant macroscopic measures are preserved, any time step is as good or as bad as any other. However, a good rule of thumb is that it should be possible to discriminate spike times an order of magnitude more accurately than the characteristic timescales of the macroscopic phenomena to be observed, such as the temporal structure of cross-correlations. Note that even if the characteristics of the system to be investigated suggest that a grid-constrained implementation is optimal, the availability of equivalent continuous-time implementations is still advantageous: should the suspicion arise that an observed phenomenon is an artifact of the grid constraints, they can be used to test this without altering any other part of the simulation. For much the same reason, it is very useful to be able to modify the time step without having to adjust the rest of the simulation.

The network studied in section 5.4 illustrates two more important points. First, it demonstrates that exceedingly small single-neuron integration errors may be required to accurately capture network synchronization. Second, it is clear from Figure 7C that continuous-time implementations are better at rendering macroscopic measures such as synchrony correctly: in conditions where the grid-constrained and prescient implementations

achieve the same single-neuron spike-timing error, the prescient implementation yields a significantly smaller error in network synchrony.

Accuracy cannot be improved on indefinitely: Figures 3 and 4 show that the errors in both membrane potential and spike timing saturate at about 10^{-14} mV and ms, respectively, for a time step of around 10^{-3} ms. The saturation accuracy is close to the maximal precision of the standard floating-point representation of the computer, and so 10^{-3} ms represents a lower bound on the range of useful h. An upper bound is determined by the physical properties of the system. First, h may not be larger than the minimum propagation delay in the network or the refractory period of the neurons. Second, using a large h increases the danger that a spike is missed. This can occur if the true trajectory of the membrane potential passes through the threshold within a step but is subthreshold again by the end of the step. This is less of an issue for the canonical implementation, as the check for a suprathreshold membrane potential is not just performed at the end of every step but also at the arrival of every incoming event (see section 4.1.1).

There is a common perception that event-driven algorithms are exact and time-driven algorithms are approximate. However, both parts of this perception are generally false. With respect to the first part, event-driven algorithms are not by the nature of the algorithm more exact than time-driven algorithms. It depends on the dynamics of the neuron model whether an event-driven algorithm can find an exact solution, just as it does for time-driven algorithms. For a restricted class of models, the spike times can be calculated exactly through inversion of the dynamics. For other models, approximate methods to determine the spike times need to be employed. With respect to the second part, time-driven algorithms are not necessarily approximate. A discrete-time algorithm does not imply that spike times have to be constrained onto the grid, as shown by Hansel et al. (1998) and Shelley and Tao (2001). Moreover, the subthreshold dynamics for a large class of neuron models can be integrated exactly (Rotter & Diesmann, 1999). Here we combine these insights to show that the degree of approximation in a simulation is not determined by whether an event-driven or a time-driven algorithm is used but by the dynamics of the neuron model.

A further question is whether the terms *time-driven* and *event-driven* should even be used in this mutually exclusive way. In our algorithm, neuron implementations treating incoming and outgoing spikes in continuous time are seamlessly integrated into a global discrete-time algorithm. Should this therefore be considered a time-driven or an event-driven algorithm? We believe that this combination of techniques represents a hybrid algorithm that is globally time driven but locally event driven. Similarly, when designing a distributed simulation algorithm (Morrison, Mehring, et al., 2005), it was shown that a time-driven neuron updating algorithm can be successfully combined with event-driven synapse updating, again suggesting that no dogmatic distinction between the two approaches need

be made. However, although we were able to demonstrate the potential advantages of a hybrid algorithm, these findings do not in principle rule out the existence of pure event-driven or time-driven algorithms with identical universality and better performance for a given set of parameters than the schemes presented here.

In closing, we express our hope that this work can help defuse the at times heated debate between advocates of event-driven and time-driven algorithms for the simulation of neuronal networks.

Appendix: Numerical Techniques

In this appendix, we present the numerical techniques employed to achieve the reported accuracy.

A.1 Accuracy of Floating-Point Representation of Physical Quantities. The `double` representation of floating-point numbers (Press, Teukolsky, Vetterling, & Flannery, 1992) used by standard computer hardware limits the accuracy with which physical quantities can be stored. The machine precision ε is the smallest number for which the `double` representation of $1 + \varepsilon$ is different from the representation of 1. Consequently, the absolute error σ_x of a quantity x is limited by the magnitude of x,

$$\sigma_x \approx 2^{\lfloor \log_2 x \rfloor} \cdot \varepsilon.$$

In `double` representation, we have

$$\varepsilon = 2^{-52} \approx 2.22 \cdot 10^{-16}.$$

Membrane potential values y are on the order of 20 mV; the lower limit of the integration error in the membrane potential Δ is therefore on the order of $5 \cdot 10^{-15}$ mV. According to the rules of error propagation, the error in the time of threshold crossing σ depends on the error in membrane potential as $\sigma = |1/\dot{y}|\Delta$. Typical values of the derivative $|\dot{y}|$ of the membrane potential are on the order of 1 mV per ms (single-neuron simulations, data not shown), from which we obtain $5 \cdot 10^{-15}$ ms as a lower bound for the error in spike timing. Therefore, the observed integration errors at which the simulations saturate are close to the limits imposed by the `double` representation for both physical quantities.

A.2 Representation of Spike Times. We have seen in section A.1 that the absolute error σ_x depends on the magnitude of the quantity x. As a consequence, the error of spike times recorded in `double` representation increases with simulation time. An additional error is introduced if the computation time step h cannot be represented as a `double` (e.g., 0.1 ms).

Therefore, we record spike times as a pair of two values $\{t + h, \delta\}$. The first one is an integral number in units of h represented as a `long int` specifying the computation step in which the spike was emitted. The second one is the offset of the spike time in units of ms represented as a `double`. If h is a power of 2 in units of ms, both values can be represented as `doubles` without loss of accuracy.

A.3 Evaluating the Update Equation. The implementation of the update equation 3.1 in the target programming language requires attention to numerical detail if utmost precision is desired. We were able to reduce membrane potential errors in a nonspiking simulation from some 10^{-12} mV to about 10^{-15} mV by careful rearrangement of terms. Although details may depend significantly on processor architecture and compiler optimization strategies, we will briefly recount the implementation we found optimal.

The matrix-vector multiplication in equation 3.1 describes updates of the form

$$y \leftarrow (1 - e^{-\frac{h}{\tau}})a + e^{-\frac{h}{\tau}} y,$$

where a and y are of order unity, while $h \ll \tau$ so that $e^{-\frac{h}{\tau}} \approx 1$. For a time step of $h = 2^{-12}$ ms and a time constant of $\tau = 10$ ms, one has $\gamma = 1 - e^{-\frac{h}{\tau}} \approx 10^{-5}$. The quantity γ can be computed accurately for small values of h/τ using the function `expm1(x)` provided by current numeric libraries (C standard library; see also Galassi et al., 2001). Using `double` resolution, γ will have some 15 significant digits, spanning roughly from 10^{-5} to 10^{-20}, and all of these digits are nontrivial. The exponential $e^{-\frac{h}{\tau}}$ may be computed to 15 significant digits using `exp(x)`; the first five of these will be trivial nines, though, leaving just 10 nontrivial digits, which furthermore span down to only 10^{-15}. We thus rewrite the equation above entirely in terms of γ,

$$y \leftarrow \gamma a + (1 - \gamma)y,$$

and finally as

$$y \leftarrow \gamma(a - y) + y.$$

In our experience, this final form yields the most accurate results, as the full precision of γ is retained as long as possible. Note that computing the $1 - \gamma$ term above discards the five least significant digits of γ.

When several terms need to be added, they should be organized according to their expected magnitude starting with the smallest components.

A.4 Polynomial Interpolation of the Membrane Potential. In order to approximate the time of threshold crossing, the membrane potential y_t^3

known at grid points t with spacing h can be interpolated with polynomials of different order. For the purpose of this section, we drop the index specifying the membrane potential as the third component of the state vector. Without loss of generality, we assume that the threshold crossing occurs at time δ^* in the interval $(0, h]$. The corresponding values of the membrane potential are denoted $y_0 < \Theta$ and $y_h \geq \Theta$, respectively. The threshold crossings are found using the explicit formulas for the roots of polynomials of order $n = 1$, 2, and 3 (Weisstein, 1999). In order to constrain the polynomials, we exploit the fact that the derivative of the membrane potential can be easily obtained from the state vector at both sides of the interval. For the grid-constrained ($n = 0$) simulation and linear ($n = 1$) interpolation, we demonstrate why the error in spike timing decreases with h^{n+1}.

A.4.1 Grid-Constrained Simulation. In the variables defined above, the approximate time of threshold crossing δ equals the computation time step h; the spike is reported to occur at the right border of the interval $(0, h]$. Assuming the membrane potential $y(t)$ to be exact, the error in membrane potential Δ with respect to the value at the exact point of threshold crossing is

$$\Delta = y_h - y(\delta^*).$$

Let us require that $y(t)$ is sufficiently often differentiable and that the derivatives assume finite values. We can then express the membrane potential as a Taylor expansion originating at the left border of the interval $y(t) = y_0 + \dot{y}_0 t + O(t^2)$. Considering terms up to first order, we obtain

$$\begin{aligned}\Delta &= \{y_0 + \dot{y}_0 h\} - \{y_0 + \dot{y}_0 \delta^*\} \\ &= \dot{y}_0 (h - \delta^*).\end{aligned}$$

Hence, Δ reaches its maximum amplitude at $\delta^* = 0$, and we can write

$$|\Delta| \leq |\dot{y}_0| h.$$

The error in spike timing is

$$\sigma = h - \delta^* = \frac{1}{\dot{y}_0} \Delta$$

and

$$|\sigma| \leq h.$$

A.4.2 Linear Interpolation. A polynomial of order 1 is naturally constrained by the values of the membrane potential (y_0 and y_h) at both ends of the interval. With $y_t = a\,t + b$, the set of equations specifying the coefficients of the polynomial (a and b) is

$$\begin{aligned}\begin{bmatrix} a \\ b \end{bmatrix} &= \begin{bmatrix} 0 & 1 \\ h & 1 \end{bmatrix}^{-1} \begin{bmatrix} y_0 \\ y_h \end{bmatrix} \\ &= \begin{bmatrix} -h^{-1} & h^{-1} \\ 1 & 0 \end{bmatrix} \begin{bmatrix} y_0 \\ y_h \end{bmatrix} \\ &= \begin{bmatrix} (y_h - y_0)h^{-1} \\ y_0 \end{bmatrix}.\end{aligned}$$

Thus, in normalized form, we need to solve

$$0 = \frac{y_h - y_0}{h}\delta + (y_0 - \Theta) \tag{A.1}$$

for δ to find the approximate time of threshold crossing. At the exact point of threshold crossing δ^*, the error in membrane potential is

$$\Delta = \left\{ \frac{y_h - y_0}{h}\delta^* + y_0 \right\} - y(\delta^*). \tag{A.2}$$

Let us require that $y(t)$ is sufficiently often differentiable and that the derivatives assume finite values. We can then express the membrane potential as a Taylor expansion originating at the left border of the interval $y(t) = y_0 + \dot{y}_0 t + \frac{1}{2}\ddot{y}_0 t^2 + O(t^3)$. Considering terms up to second order, we obtain

$$\begin{aligned}\Delta &= \left\{ \dot{y}_0\delta^* + \frac{1}{2}\ddot{y}_0 h\delta^* + y_0 \right\} - \left\{ y_0 + \dot{y}_0\delta^* + \frac{1}{2}\ddot{y}_0\delta^{*^2} \right\} \\ &= \frac{1}{2}\ddot{y}_0(\delta^* h - \delta^{*^2}).\end{aligned}$$

The time of threshold crossing is bounded by the interval $(0, h]$. Hence, Δ reaches its maximum amplitude at $\delta^* = \frac{1}{2}h$, and we can write

$$|\Delta| \leq \frac{1}{8}\,|\ddot{y}_0|\,h^2.$$

Noting that $y(\delta^*) = \Theta$ in equation A.2, we have

$$\Delta = \frac{y_h - y_0}{h}\delta^* + (y_0 - \Theta), \tag{A.3}$$

and subtracting equation 4.1 from 4.3, we obtain

$$\Delta = \frac{y_h - y_0}{h}(\delta^* - \delta).$$

Thus, the error in spike timing is

$$\sigma = \delta^* - \delta = \frac{h}{y_h - y_0}\Delta.$$

With the help of the expansion

$$\frac{h}{y_h - y_0} = \frac{1}{\dot{y}_0} - \frac{\ddot{y}_0}{2\dot{y}_0^2}h,$$

we arrive at

$$|\sigma| \leq \frac{1}{8}\left|\frac{\ddot{y}_0}{\dot{y}_0}\right|h^2.$$

A.4.3 Quadratic Interpolation. Using a polynomial of order 2, we can add an additional constraint to the interpolating function. We decide for the derivative of the membrane potential at the left border of the interval $\dot{y}_0$. With $y_t = at^2 + bt + c$, we have

$$\begin{aligned}
\begin{bmatrix} a \\ b \\ c \end{bmatrix} &= \begin{bmatrix} 0 & 0 & 1 \\ h^2 & h & 1 \\ 0 & 1 & 0 \end{bmatrix}^{-1} \begin{bmatrix} y_0 \\ y_h \\ \dot{y}_0 \end{bmatrix} \\
&= \begin{bmatrix} -h^{-2} & h^{-2} & h^{-1} \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} y_0 \\ y_h \\ \dot{y}_0 \end{bmatrix} \\
&= \begin{bmatrix} (y_h - y_0)h^{-2} - \dot{y}_0 h^{-1} \\ \dot{y}_0 \\ y_0 \end{bmatrix}.
\end{aligned}$$

Thus, in normalized form, we need to solve

$$0 = \delta^2 + \frac{\dot{y}_0 h^2}{y_h - y_0 - \dot{y}_0 h}\delta + \frac{(y_0 - \Theta)h^2}{y_h - y_0 - \dot{y}_0 h}.$$

The solution can be obtained by the quadratic formula. The GSL (Galassi et al., 2001) implements an appropriate solver. Generally there are two real solutions: the desired one inside the interval $(0, h]$ and one outside.

A.4.4 Cubic Interpolation. A polynomial of order 3 enables us to constrain the interpolation further by the derivative of the membrane potential at the right border of the interval $\dot{y}_h$. With $y_t = at^3 + bt^2 + ct + d$, we have

$$\begin{aligned}
\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} &= \begin{bmatrix} 0 & 0 & 0 & 1 \\ h^3 & h^2 & h & 1 \\ 0 & 0 & 1 & 0 \\ 3h^2 & 2h & 1 & 0 \end{bmatrix}^{-1} \begin{bmatrix} y_0 \\ y_h \\ \dot{y}_0 \\ \dot{y}_h \end{bmatrix} \\
&= \begin{bmatrix} 2h & -2h^{-3} & h^{-2} & h^{-2} \\ -3h^{-2} & 3h^{-2} & -2h^{-1} & -h^{-1} \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} y_0 \\ y_h \\ \dot{y}_0 \\ \dot{y}_h \end{bmatrix} \\
&= \begin{bmatrix} 2(y_0 - y_h)h^{-3} + (\dot{y}_0 + \dot{y}_h)h^{-2} \\ 3(y_h - y_0)h^{-2} - (2\dot{y}_0 + \dot{y}_h)h^{-1} \\ \dot{y}_0 \\ y_0 \end{bmatrix}.
\end{aligned}$$

Thus, in normalized form, we need to solve

$$\begin{aligned}
0 = \delta^3 &+ \frac{3(y_h - y_0)h - (2\dot{y}_0 + \dot{y}_h)h^2}{2(y_0 - y_h) + (\dot{y}_0 + \dot{y}_h)h}\delta^2 \\
&+ \frac{\dot{y}_0 h^3}{2(y_0 - y_h) + (\dot{y}_0 + \dot{y}_h)h}\delta + \frac{(y_0 - \Theta)h^3}{2(y_0 - y_h) + (\dot{y}_0 + \dot{y}_h)h}.
\end{aligned}$$

The solution can be found by the cubic formula. There is at least one real solution in the interval $(0, h]$. It is convenient to chose a substitution that avoids the intermediate occurrence of complex quantities (e.g., Weisstein, 1999). The GSL (Galassi et al., 2001) implements an appropriate solver. If the interval contains more than one solution, the time of threshold crossing is defined by the left-most root.

Acknowledgments

The new address of A. M. and M. D. is Computational Neuroscience Group, RIKEN Brain Science Institute, Wako, Japan. The new address of S.S. is Human-Neurobiologie, University of Bremen, Germany. We acknowledge Johan Hake for the interesting discussions that started the project.

As always, Stefan Rotter and the members of the NEST collaboration (in particular Marc-Oliver Gewaltig) were very helpful. We thank Jochen Eppler for assisting with the C++ coding. We acknowledge the two anonymous referees whose stimulating questions helped us to improve the letter. This work was partially funded by DAAD 313-PPP-N4-lk, DIP F1.2, BMBF Grant 01GQ0420 to the Bernstein Center for Computational Neuroscience Freiburg, and EU Grant 15879 (FACETS). As of the date this letter is published, the NEST initiative (www.nest-initiative.org) makes available the different implementations of the neuron model used as an example in this article, in source code form under its public license. All simulations were carried out using the parallel computing facilities of the Norwegian University of Life Sciences at Ås.

References

Brette, R. (2006). Exact simulation of integrate-and-fire models with synaptic conductances. *Neural Comput., 18*, 2004–2027.

Brunel, N. (2000). Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons. *J. Comput. Neurosci., 8*(3), 183–208.

Diesmann, M., & Gewaltig, M.-O. (2002). NEST: An environment for neural systems simulations. In T. Plesser & V. Macho (Eds.), *Forschung und wisschenschaftliches Rechnen, Beiträge zum Heinz-Billing-Preis 2001* (pp. 43–70). Göttingen: Gesellschaft für wissenschaftliche Datenverarbeitung.

Diesmann, M., Gewaltig, M.-O., Rotter, S., & Aertsen, A. (2001). State space analysis of synchronous spiking in cortical neural networks. *Neurocomputing, 38–40*, 565–571.

Ferscha, A. (1996). Parallel and distributed simulation of discrete event systems. In A. Y. Zomaya (Ed.), *Parallel and distributed computing handbook* (pp. 1003–1041). New York: McGraw-Hill.

Fujimoto, R. M. (2000). *Parallel and distributed simulation systems*. New York: Wiley.

Galassi, M., Davies, J., Theiler, J., Gough, B., Jungman, G., Booth, M., & Rossi, F. (2001). *Gnu scientific library: Reference manual*. Bristol: Network Theory Limited.

Golub, G. H., & van Loan, C. F. (1996). *Matrix computations* (3rd ed.). Baltimore, MD: Johns Hopkins University Press.

Hammarlund, P., & Ekeberg, O. (1998). Large neural network simulations on multiple hardware platforms. *J. Comput. Neurosci., 5*(4), 443–459.

Hansel, D., Mato, G., Meunier, C., & Neltner, L. (1998). On numerical simulations of integrate-and-fire neural networks. *Neural Comput., 10*(2), 467–483.

Harris, J., Baurick, J., Frye, J., King, J., Ballew, M., Goodman, P., & Drewes, R. (2003). *A novel parallel hardware and software solution for a large-scale biologically realistic cortical simulation* (Tech. Rep.). Las Vegas: University of Nevada.

Heck, A. (2003). *Introduction to Maple* (3rd ed.). Berlin: Springer-Verlag.

Lytton, W. W., & Hines, M. L. (2005). Independent variable time-step integration of individual neurons for network simulations. *Neural Comput., 17*, 903–921.

Makino, T. (2003). A discrete-event neural network simulator for general neuron models. *Neural Comput. and Applic., 11*, 210–223.

Marian, I., Reilly, R. G., & Mackey, D. (2002). Efficient event-driven simulation of spiking neural networks. In *Proceedings of the 3. WSES International Conference on Neural Networks and Applications*. Interlaken, Switzerland.

Mattia, M., & Del Giudice, P. (2000). Efficient event-driven simulation of large networks of spiking neurons and dynamical synapses. *Neural Comput., 12*(10), 2305–2329.

Mirollo, R. E., & Strogatz, S. H. (1990). Synchronization of pulse-coupled biological oscillators. *SIAM J. Appl. Math., 50*(6), 1645–1662.

Morrison, A., Hake, J., Straube, S., Plesser, H. E., & Diesmann, M. (2005). Precise spike timing with exact subthreshold integration in discrete time network simulations. Proceedings of the 30th Göttingen Neurobiology Conference. *Neuroforum, 1*(Suppl.), 205B.

Morrison, A., Mehring, C., Geisel, T., Aertsen, A., & Diesmann, M. (2005). Advancing the boundaries of high connectivity network simulation with distributed computing. *Neural Comput., 17*(8), 1776–1801.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (1992). *Numerical recipes in C* (2nd ed.). Cambridge: Cambridge University Press.

Reutimann, J., Giugliano, M., & Fusi, S. (2003). Event-driven simulation of spiking neurons with stochastic dynamics. *Neural Comput., 15*, 811–830.

Rochel, O., & Martinez, D. (2003). An event-driven framework for the simulation of networks of spiking neurons. In *ESANN'2003 Proceedings—European Symposium on Artifical Neural Networks* (pp. 295–300). Bruges, Belgium: d-side Publications.

Rotter, S., & Diesmann, M. (1999). Exact digital simulation of time-invariant linear systems with applications to neuronal modeling. *Biol. Cybern., 81*(5/6), 381–402.

Shelley, M. J., & Tao, L. (2001). Efficient and accurate time-stepping schemes for integrate-and-fire neuronal networks. *J. Comput. Neurosci., 11*(2), 111–119.

Sloot, A., Kaandorp, J. A., Hoekstra, G., & Overeinder, B. J. (1999). Distributed simulation with cellular automata: Architecture and applications. In J. Pavelka, G. Tel, & M. Bartosek (Eds.), *SOFSEM'99*, LNCS (pp. 203–248). Berlin: Springer-Verlag.

Tuckwell, H. C. (1988). *Introduction to theoretical neurobiology*. Cambridge: Cambridge University Press.

Weisstein, E. W. (1999). *CRC concise encyclopedia of mathematics*. Boca Raton, FL: CRC Press.

Wolfram, S. (2003). *The mathematica book* (5th ed.). Champaign, IL: Wolfram Media Incorporated.

Zeigler, B. P., Praehofer, H., & Kim, T. G. (2000). *Theory of modeling and simulation: Integrating discrete event and continuous complex dynamic systems* (2nd ed.). Amsterdam: Academic Press.

Received August 12, 2005; accepted May 25, 2006.