German Research School
for Simulation Sciences

German Research School for Simulation Sciences

Detection of threshold crossings in the leaky integrate-and-fire neuron model with α-shaped postsynaptic currents in time-driven simulations

Master's Thesis

Jeyashree Krishnan

communicated by Univ.-Prof. Marek Behr

Thursday 16th October, 2014

Supervisor
Dr.Moritz Helias

Examiner
Prof.Dr.Markus Diesmann

Co-Examiner
Dr.Edoardo Di Napoli

Contents

1 Introduction

Neuroscience is the scientific study of the nervous system that attempts to understand information processing in the brain in terms of electrical activity in different parts of the central nervous system. A neuron, the functional unit of the nervous system, is an electrically excitable cell that processes and transmits information through electrical and chemical signals via the points of contacts, called synapses. The brain is a complex system with high connectivity. Neural systems are nonlinear and stochastic in nature. Experimental data to study such a system is only available either for single or extremely small numbers of neurons or for superposition signals from very large populations, necessitating the application of theoretical and computational methods to understand the dynamic properties of neurons. Therefore, neural network simulations are crucial for the advancement of brain research.

Computational neuroscience is the study of brain function using theoretical methods and computer simulations. Simulations of neurons employ mathematical models to focus on reproducing the neural activity in the brain. The dynamics of a neuron can be described by a set of differential equations where the interaction between a pair of neurons is simulated as point events, i.e. spikes. The spike or action potential is the unit of signal transmission. A typical neuron possesses a cell body, dendrites, and an axon. The dendrites act as input devices and collect input signals from other neurons and transmit them to the soma. The cell body can be considered as the central processing unit that performs a non-linear processing step i.e. if the input exceeds a certain threshold voltage then a spike occurs. The axon or the output device delivers the output signal to other neurons. By convention, the sending neuron is referred to as the presynaptic cell and the receiving neuron as the postsynaptic cell. A spike causes a small change in the membrane potential of the target neurons, called the postsynaptic potential (PSP). These PSPs can then further initiate or inhibit action potentials.

The membrane potential can be static, in absence of inputs, or dynamic, if the cell receives synaptic input. When synaptic inputs are absent, the membrane potential of the neuron assumes a static value, called the resting voltage. This potential difference between the interior and exterior of a neuron is determined by differences of ionic concentrations inside and outside the cell body, that are kept up by active transport mechanisms.

Two common approaches to modeling the nervous system are the top-down and bottom-up approaches. In the bottom-up approach neuron are considered as nodes and their synaptic connections as edges. The network can then be described in terms of the neurons, and their projections. This approach is referred to as bottom-up, since we construct the network from its basic components, the neurons. Hence there are simulation tools, techniques for single or small numbers of neurons that focus on their morphology and function, like the NEURON simulator [Hines, 1993], and simulation tools to study large networks of simple cells, like SPLIT [Djurfeldt, 2009], the Brian simulator [Goodman and Brette, 2009], the

C2 simulator [Ananthanarayanan and Modha, 2007], and NEST [Gewaltig and Diesmann, 2007]. In a top-down approach, an algorithm described at an abstract mathematical level is implemented using the constituents available in a neuronal network, i.e. neurons and synapses.

Simplified neuron models that neglect the morphological structure of the individual cells, enables the simulation of a large number of neurons while maintaining an acceptable degree of biological detail [Diesmann and Gewaltig, 2002]. Neural network simulators are software applications that are used to simulate the behavior of artificial or biological neural networks. NEST is a simulator for networks of point neurons, that is, neuron models that collapse the morphology of different parts of a neuron to a single compartment or a small number of compartments [Gewaltig and Diesmann, 2007]. NEST considers neurons as nodes with possibly complex connectivity and is aimed to understand the dynamics of large neuronal networks.

Two classical approaches to simulate neuronal networks are the time-driven and event-driven methods [Fujimoto, 2000, Zeigler et al., 2000, Sloot et al., 1999, Ferscha, 1996]. Both of these methods describe the physical system in terms of a set of state variables representing the neurons and events mediate the interaction between cells. In a time-driven algorithm the time evolution advances the dynamics of each neuron in time steps defined by a computational step size h as illustrated in Figure 2.0.2 on page 14. The system is monitored in unitary time intervals. The characteristics of a time-driven simulation are a fixed-size simulation step and a fixed-size communication interval. The simulation step size determines discrete points in time when all neurons are updated and checked for their membrane potential to cross the threshold. The choice of this constant size is, as in any numerical simulation, a trade-off between precision and simulation time. In the implementation with precise spike timing [Morrison et al., 2007b], a variant of the time-driven approach, the arrival times of incoming spikes introduce additional update-and-check points. Incoming spikes are incorporated and membrane potential crossings are detected only at these update-and-check points. The communication interval is a multiple of the simulation step size and defines those discrete points where neurons communicate their spikes to other neurons. This interval can be as long as the minimum synaptic delay in the network [Morrison et al., 2005b]. If the voltage of one neuron crosses the threshold, then a spike is delivered to each of the neurons that it is connected to. After all the neurons are updated, the next iteration begins. The simulation step can therefore be increased up to the size of the communication interval. The detection of a threshold crossing can only take place at a check point, but the timing of the spike is estimated with precision limited only by the representation of the double floating point number, ϵ [Kunkel et al., 2011], where ϵ is the smallest number for which the double representation of $1 + \epsilon$ is different from 1 [Morrison et al., 2007b].

In the event-driven simulation, we observe the system only at points in time where spike occurrence or a threshold crossing is detected when a neuron receives an event. If this event leads to a spike occurrence, then it is inserted into the queue. Future event occurrences induced by states have to be scheduled. Apart from the global clock and state variables indicating the current state of the system that is also characteristic to a time-driven simu-

lation, an event-driven simulation also maintains a time ordered event list for those events in the queue [Ferscha, 1996]. Neuron models may exhibit invertible or non-invertible dynamics. If, for example, the arrival of a spike causes an immediate jump in the membrane potential followed by an exponential decay this is an invertible dynamics; a spike can only be triggered at the point of arrival of an excitatory synaptic event. If the arrival of a spike causes an excursion of the membrane potential only within a certain rise time, the situation becomes more complicated. Even though the time point of threshold crossing due to the incoming event can be predicted, it may have to be corrected, if further events arrive at the neuron [Morrison and Diesmann, 2008].Consider a network of 10^5 neurons with a computation step size of 0.1 ms. If each neuron is receiving inputs at an average rate of 1 Hz from each of the 10^4 synapses it needs to process a spike at approximately every time step,meaning 10^9 neuron updates are performed per second. Considering these two factors: non-invertible dynamics and simulation time, time-driven simulations are generally preferred. Such a scheme can incorporate any kind of subthreshold dynamics without making changes to the update and spike delivery algorithm and has the same computational cost as using an event-driven framework when the integration error is taken as the metric of performance, instead of time interval [Morrison et al., 2007a]. Therefore one is normally inclined to favor a time-driven framework for the simulation of large, highly connected networks. This dynamics can then be propagated in continuous-time or discrete-time. In a discrete time simulation the occurrence of spikes are constrained to the grid. Forcing spikes on the grid can however distort the synchronization dynamics of some networks [Hansel et al., 1998]. Hence a continuous-time algorithm is generally preferred.

The leaky integrate-and-fire (LIF) neuron model is one of the commonly used spiking neuron models that can mimic the dynamics of neurons to high accuracy [Rauch et al., 2003]. These models consist of a system of first order linear differential equations, where the sub-threshold dynamics can be exactly integrated [Rotter and Diesmann, 1999] and any excursions that lead to the membrane potential to cross a threshold lead to the emission of a spike. Kunkel et al. [2011] presented a time-driven simulation method to handle off-grid spiking in combination with exact sub-threshold integration of LIF neuron models with exponential PSCs. The spikes are processed sequentially within a predefined time step and then the dynamics is propagated from spike to spike till the end of the time step. If, in between the time step there occurs a supra-threshold value of the membrane potential the time point of threshold crossing can be calculated by interpolation. The state of each neuron is updated on an equally spaced time grid and the spikes are detected by comparison of the neuronal state before and after the update. Here spikes are represented by an integer time stamp and an offset δ. For spiking neurons, a popular model for the postsynaptic currents is the α-function. Neuron models with ionic currents exhibit a double exponential membrane dynamics. The α-shaped PSCs are a good approximation of this, and so it is a frequently used model to describe the time course of ionic currents [Bernard et al., 1994, Wilson and Bower, 1989]. Using such a model that has finite rise time reduces artificial synchrony. In the implementation with precise spike timing, the membrane potential is in addition propagated and checked at the points in time when a spike from another neuron is received. Typically in a time-driven simulation spikes are detected only at the predefined

update-and-check points. In the case of a brief membrane potential excursion a threshold crossing might therefore not be detected at the next check point.

Morrison et al. [2007a] developed the canonical α-model that handles events by exact subthreshold integration in discrete-time neural networks with continuous spike times. Hansel et al. [1998] and Shelley and Tao [2001] discuss the method of a precise simulation wherein the exact spike timing is calculated by interpolating the spike time within the grid. Morrison et al. [2005a] addresses why one needs to adopt the precise spike timing approach. In a purely on-grid scenario, the threshold crossings that occurs between two consecutive intervals of a time grid and would not carry a precise timestamp which may lead to artificial synchronization. In addition, grid-constrained spiking causes an integration error that declines only linearly with the resolution h. The idea of the current work is to extend this existing α-model by including a spike test as implemented by Kunkel et al. [2011]. Supplementing the canonical α-model in NEST with standard tests that checks for supra-threshold membrane potential using an algorithm that can compute the peak time of threshold crossing between two consecutive interval warrants that the new model (called "lossless" in the following) is a precise implementation of the mathematical definition of the model. The objective of this work therefore is to implement a series of tests based on the initial and final conditions of the dynamic variables at the beginning and the end of a simulation time step that allow to detect whether or not a spike has been missed. Spike misses can be traced back in time by expressing the peak time in terms of the Lambert W function [Corless et al., 1996]. These tests determine whether the initial conditions at the left check point and the final conditions at the right check point indicate that an excursion exceeding the threshold has happened in between. We need to find a computationally cheap and efficient implementations of the spike tests to catch the missed spikes given the typical statistics of input state variables. The resulting missed spike can be detected by including this spike-test algorithm at all check points.

2 Subthreshold dynamics in time-driven simulations

The leaky integrate-and-fire neuron is the simplest and one of the best known spiking neuron models due to the ease with which it can be simulated and analyzed [Plesser and Diesmann, 2009]. In its simplest form, the neuron is modeled as a "leaky integrator" of its input $I(t)$

$$\frac{dV}{dt} = -\frac{1}{\tau_m}V(t) + \frac{1}{C}I(t), \tag{2.0.1}$$

where $V(t)$ represents the membrane potential at time t, $\tau_m = RC$ is the membrane time constant, R is the membrane resistance and C is the capacitance. This equation describes a simple resistor-capacitor (RC) circuit, where the leakage term is due to the resistor and the integration of $I(t)$ is due to the capacitor that is connected in parallel to the resistor. The action potential is not explicitly modeled in the LIF model. Instead, when the membrane potential $V(t)$ reaches the spiking threshold Θ it is instantaneously set to a reset potential V_r and this is followed by a delay or a momentary refractory period τ_r immediately after $V(t)$ is set to V_r.

The α-shaped postsynaptic current (α-PSC) elicited upon arrival of a synaptic impulse at $t = 0$ is given by

$$I_\alpha(t) = \hat{i}\frac{e}{\tau_\alpha} t e^{\frac{-t}{\tau_\alpha}}, \tag{2.0.2}$$

where $\hat{i}$ is the peak value of the current and τ_α is the rise time. The values of these parameters used in this work are $\tau_m = 10.0$ ms, $\tau_\alpha = 2.0$ ms, $C = 250.0$ pF, $\Theta = 20$ mV . The dynamics of the neuron model is linear and can be formulated into a system of equations as in Rotter and Diesmann [1999], who combine the state variable in a three component vector

$$\begin{aligned} y^1 &= \frac{dI}{dt} + \frac{I}{\tau_\alpha} \\ y^2 &= I \\ y^3 &= V. \end{aligned} \tag{2.0.3}$$

The system is propagated in time by the equation

$$\dot{y} = Ay \tag{2.0.4}$$

with the coefficient matrix

$$A = \begin{bmatrix} -\frac{1}{\tau_\alpha} & 0 & 0 \\ 1 & -\frac{1}{\tau_\alpha} & 0 \\ 0 & \frac{1}{C} & -\frac{1}{\tau_m} \end{bmatrix}, \tag{2.0.5}$$

where the first two components of y describe the sub-system generating the PSCs (2.0.2) and the third component is the membrane potential. The initial condition of the system of differential equations (2.0.4) for the arrival of a single synaptic impulse is

$$y(0) = \begin{bmatrix} \hat{i}\frac{e}{\tau_\alpha} \\ 0 \\ 0 \end{bmatrix}, \tag{2.0.6}$$

where $y(0)$ is the initial condition for a postsynaptic potential starting at time $t = 0$. Here $\frac{e}{\tau_\alpha}$ is the scaling factor, ensuring that the posystynaptic current has $\hat{i}$ as the peak value. The exact solution for this system is given by

$$y(t) = P(t)\, y(0). \tag{2.0.7}$$

In a temporal grid $t = hk$ the propagator matrix given by

$$P(h) = e^{At}, \tag{2.0.8}$$

which is a matrix exponential that propagates the system from one grid point to another by

$$y_{t+h} = P(h)\, y_t. \tag{2.0.9}$$

$P(h)$ can be obtained in closed form and, for a discrete time step h, needs to be calculated only once at the beginning of the simulation making it an efficient method. The propagator takes the explicit form

$$P(h) = \begin{pmatrix} e^{-\frac{h}{\tau_\alpha}} & 0 & 0 \\ he^{-\frac{h}{\tau_\alpha}} & e^{-\frac{h}{\tau_\alpha}} & 0 \\ \frac{1}{C}\left(\frac{e^{-\frac{h}{\tau_m}} - e^{-\frac{h}{\tau_\alpha}}}{\left(\frac{1}{\tau_\alpha} - \frac{1}{\tau_m}\right)^2} - \frac{he^{-\frac{h}{\tau_\alpha}}}{\left(\frac{1}{\tau_\alpha} - \frac{1}{\tau_m}\right)}\right) & \frac{1}{C}\frac{e^{-\frac{h}{\tau_m}} - e^{-\frac{h}{\tau_\alpha}}}{\left(\frac{1}{\tau_\alpha} - \frac{1}{\tau_m}\right)} & e^{-\frac{h}{\tau_m}} \end{pmatrix}. \tag{2.0.10}$$

The description of exact integration and precise spike timing that follows here is in large parts based on Morrison et al. [2007b].

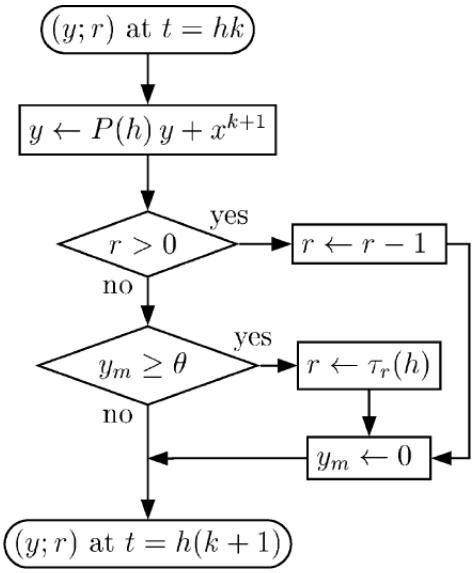


Figure 2.0.1: Flowchart illustrating the sub-threshold dynamics in a neuron, adapted from Diesmann et al. [2001]. The state of the neuron is given by the state vector y for the sub-threshold dynamics and an integer r measuring the time spent in the absolute refractory period $\tau_r(h)$. A constant propagator $P(h)$ propagates the sub-threshold dynamics in time. The state variables y_1 and y_2 describe the sub-system generating the PSCs and y_3 the membrane voltage. The flowchart represents the propogation of the state variables from time point k to $k+1$. The point of reference is the end of the time interval. First the state vector is updated by applying the propagator matrix to it. If the neuron is not refractory, a check for threshold crossing is done in which the membrane voltage is set to zero and neuron enters refractorniness. If the neuron is refractory in the first place then the iterative loop runs until the neuron is not refractory, upon which test for threshold crossing is done.

The complete update step can then be written as

$$y_{t+h} = P(h) y_t + x_{t+h} \tag{2.0.11}$$

where x_{t+h} is the set of initial conditions given by

$$x_{t+h} = \begin{bmatrix} \frac{e}{\tau_\alpha} \\ 0 \\ 0 \end{bmatrix} \sum_{k \in S_{t+h}} \hat{i}_k \tag{2.0.12}$$

and 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 $\hat{i}_k$ is the weight of synapse k i.e. jump or decay of membrane potential due to the input. Spike arriving at $t = h(k+1)$ causes changes to x^{k+1} which add linearly to state.

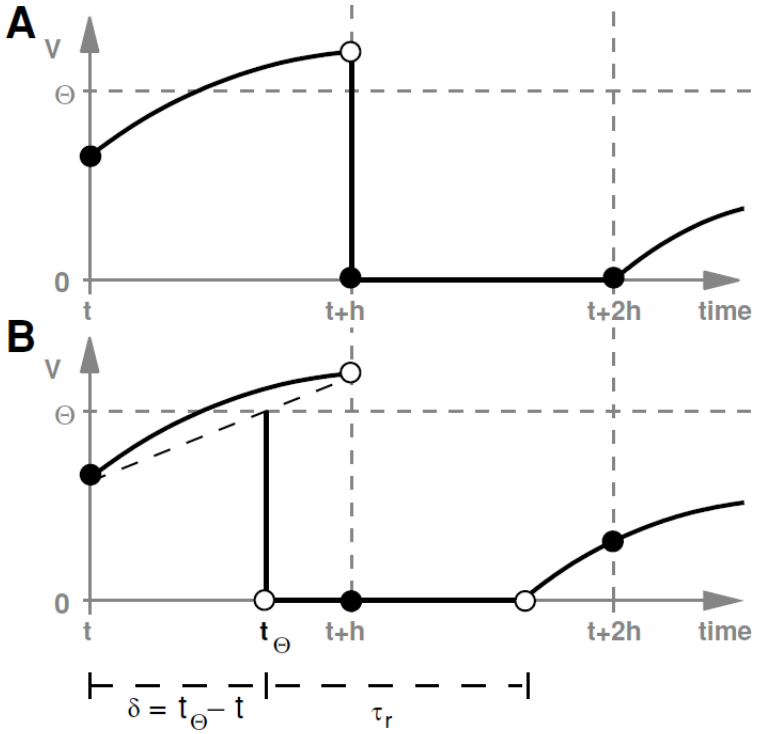


Figure 2.0.2: **(A)** Spike generation and refractory periods for conventional grid-constrained method. **(B)** Continuous time implementation (precise spike timing, adapted from Morrison et al. [2007b]). The spike threshold Θ is indicated by a horizontal dashed line. The solid black curve shows the membrane potential dynamics for a neuron that receives a suprathreshold constant input current such that there occurs a threshold crossing in the interval $(t, t+h]$. The gray vertical lines indicate the discrete time grid with spacing h. Filled circles denote observable values of the membrane potential, unfilled circles denote supporting points that are not observable. In the grid-constrained case, the spike is emitted at $t+h$ followed by a short refractory period τ_r (equivalent to grid size h). In the continuous-time implementation, the threshold crossing is found by interpolation at time $t < t_\Theta < t+h$.

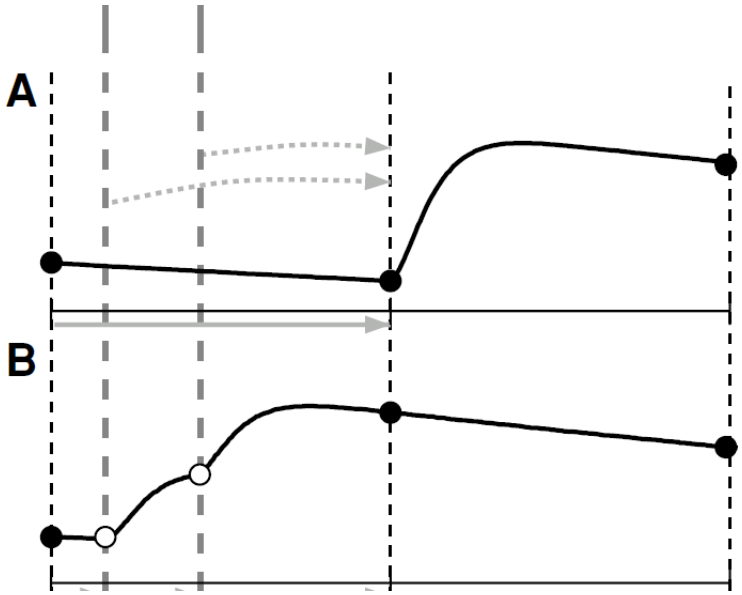


Figure 2.0.3: Handling incoming spikes: Panel **(A)** shows the grid-constrained case and **(B)** the canonical implementation. The curve represents the dynamics of membrane potential in response to two incoming spikes shown here by the vertical gray dashed lines. Filled circles denote known value of membrane potential and unfilled circles unobservable values. The gray horizontal arrows indicate the propagation steps performed during the timestep $(t, t+h]$. In the grid-constrained case, the incoming spikes are shifted to the next point in the grid which stores imprecise timestamp for the input spike which may lead to artificial synchrony. In the canonical implementation, though the membrane potential at these points are unknown, they are propagated with the precise spike timing. Figure adapted from Morrison et al. [2007b].

The time-driven simulation environment in NEST [Gewaltig and Diesmann, 2007] has an "on-grid" and "off-grid" framework that handle spikes differently. In the on-grid framework, spikes are constrained to the equidistant time grid. Here, we will not consider this mode of operation further. In the off-grid framework, spikes can be emitted at any point in time. The canonical leaky integrate-and-fire neuron model with α-shaped postsynaptic currents has been implemented by Morrison et al. [2007b] in the neural simulator NEST [Gewaltig and Diesmann, 2007] and combines the idea of exact integration and precise spike timing computed by interpolation. The precise timing of incoming events is stored in terms of their offsets δ in addition to the time steps of h in which they arrive, as illustrated in Figure 2.0.2 on page 14. Given a sorted list of event offset $\{\delta_1, \delta_2, \ldots, \delta_n\}$ with $\delta_i \leq h$, the spikes are processed at the end of every timestep, but their exact point of arrival as indicated by δ_i is taken into account. If the membrane potential y^3_{t+h} exceeds the threshold Θ, the neuron communicates a spike event to the network with a time stamp of $t+h$ and the membrane potential is set to zero in the interval $[t_\Theta, t_\Theta + \tau_r]$. The reference point is still the end of

interval, and hence the spike emitted carries the timestamp $t+h$ with an offset $\delta = t_\Theta - t$. The neuron is then refractory from t_Θ till $t_\Theta + \tau_r$ where in the membrane potential is clamped to zero and starts evolving thereafter. Given the sequence of incoming synaptic events with offsets $\{\delta_1, \delta_2, \ldots, \delta_n\}$, the subthreshold dynamics is propagrated within one time step of duration h from event to event, to the time point of arrival of the first event

$$y_{t+\delta_1} = P(\delta_1) y_t + x_{\delta_1}, \tag{2.0.13}$$

to the time point of the second event

$$y_{t+\delta_2} = P(\delta_2 - \delta_1) y_{t+\delta_1} + x_{\delta_2} \tag{2.0.14}$$

$$\vdots$$

$$y_{t+\delta_n} = P(\delta_n - \delta_{n-1}) y_{t+\delta_{n-1}} + x_{\delta_n} \tag{2.0.15}$$

till the end of the timestep

$$y_{t+h} = P(h - \delta_n) y_{t+\delta_n}. \tag{2.0.16}$$

If the membrane potential of the neuron is $y^3_{t+\delta_i} < \Theta$ and $y^3_{t+\delta_{i+1}} \geq \Theta$ then the membrane potential of the neuron reached threshold between $t+\delta_i$ and $t+\delta_{i+1}$. Since the dynamics of the neuron is non-invertible, this threshold crossing is detected by interpolation in the canonical model. This event now carries the timestamp $t+h$ and an offset $\delta = t_\Theta - t$. After spike is emitted, y_3 is reset to zero and the neuron undergoes refractoriness for a duration τ_r.

We see that the simulation step and the arrival of synaptic events defines the update-and-check points, and the communication interval defines the discrete points in time when all the neurons communicate their spikes. The communication interval is a multiple of the simulation step size and is limited only by the synaptic delay in the network [Morrison and Diesmann, 2008, Kunkel et al., 2011]. Also, the simulation step size is bounded by the size of this communication interval. Consider a situation illustrated in Figure 2.0.4 on page 17 when a short excursion of the membrane potential above threshold happens between the two check points and goes undetected, resulting in a missed spike.

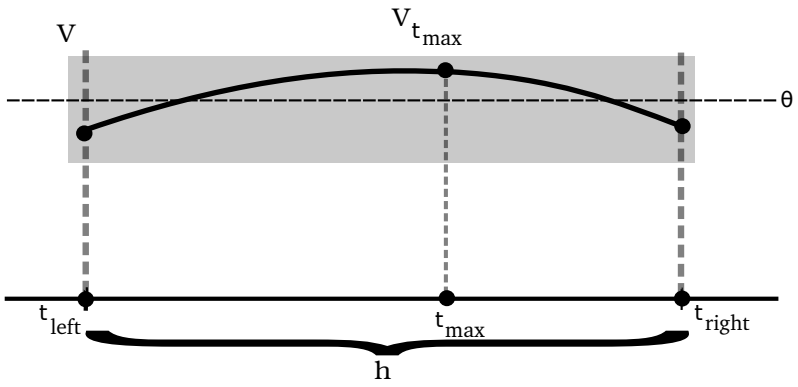


Figure 2.0.4: Illustration of a case when the threshold crossing maybe missed between two consecutive intervals on the time grid. t_{left} and t_{right} refer to the left and right ends points of the reference time interval. t_θ is the point in time when threshold crossing occurs, t_{peak} is the time at which the maximum of the membrane potential occurs. If the maximum appears between the two consecutive time intervals this may not be detected at the t_{right}.

The existing canonical α- model detects whether there has been a threshold crossing point t_Θ (where $t < t_\Theta < t + h$) by interpolation (linear, quadratic and cubic) as in Figure 2.0.2 on page 14 . However having the knowledge of the peak time would be desirable which can be computed by he new lossless α-model using the Lambert W function [Corless et al., 1996]. In the unpublished work on "*Lambert-W reveals peak time of postsynaptic potential evoked by α-shaped current*" by Markus Diesmann and Hans Ekkehard Plesser, the rationale behind deriving a closed form expression for the peak time of PSP was discussed and an analytical solution for the peak time was derived. Knowing the precise value of peak time helps compute the peak value of voltage at that point. The peak time determines the rise time of the PSP and temporal precision of neuronal response, relevant for studies of spike synchronization [Goedeke and Diesmann, 2008]. The postsynaptic potential that correspond to (2.0.2) can be efficiently represented as the solution to a system of linear differential equations. At $t = 0$ it can be obtained by solving (2.0.1) which yields

$$u(t) = \frac{e}{\tau_\alpha}\frac{1}{C}\left[\frac{e^{\frac{-t}{\tau_m}} - e^{\frac{-t}{\tau_\alpha}}}{\left(\frac{1}{\tau_\alpha} - \frac{1}{\tau_m}\right)^2} - \frac{te^{\frac{-t}{\tau_\alpha}}}{\left(\frac{1}{\tau_\alpha} - \frac{1}{\tau_m}\right)}\right], \quad \tau_m > \tau_\alpha \tag{2.0.17}$$

where the PSC is normalized to unit amplitude [Plesser and Diesmann, 2009, Wilson and Bower, 1989]. The choice $\tau_m > \tau_\alpha$ covers the PSP shapes with biologically realistic rise and decay times. $u(t)$ must be scaled to a certain amplitude because the relation of this amplitude to the spike threshold determines the interaction strength or synaptic weight.

The idea of the current work is to extend this analytics done by Diesmann (unpubl.) et al. to include all three sub-systems that constitute the propagator matrix (current and voltage)

(2.0.8) and get an explicit expression for the peak time of PSP using the Lambert W function [Corless et al., 1996]. This result can then be used to construct tests to be included in the canonical α-model of the precise spike timing framework to catch the spike misses that happen between grid points.

3 Determination of peak time using Lambert W function

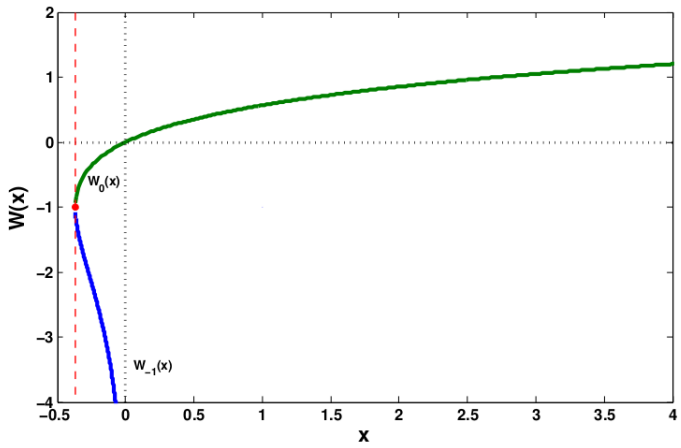
The peak time of PSP refers to the point in time where the voltage reaches its maximum. In Chapter 2 it was motivated that knowledge of the exact location of peak in the temporal space reduces artificial synchronization and integration error. The time to maximum can then be computed using the Lambert W function [Corless et al., 1996] as enumerated in the following sections of this chapter.

3.1 Lambert W function

The Lambert W function defined by $W(x)$ is the inverse of the function $f(x) = xe^x$, satisfying

$$W(x)e^{W(x)} = x \tag{3.1.1}$$

for any complex number x, and W is any complex number. Since the function is not injective, the relation W is multivalued except at zero [Corless et al., 1996].



A

B

x	$W(x)$
$x > 0$	unique real roots
$x = 0$	0
$\frac{-1}{e} < x < 0$	negative real roots :$W_0(x)$, $W_{-1}(x)$
$x = \frac{-1}{e}$	-1
$x < \frac{-1}{e}$	no real solutions

Figure 3.1.1: **(A)** Lambert W function showing the two real branches of $W(x)$. The principal branch is shown by the green curve and the non-principal branch by the blue curve. The red vertical line through $(-1/e, -1)$ marks the left border of the regime in which the two real-valued solutions exist and the red dot is the transition from one branch to the other. **(B)** Solution of equation (3.1.1) (Lambert W function) for different range of values of the argument x.

The upper branch with $W \geq -1$ is called the principal branch $W_0(x)$ and the lower branch with $W \leq -1$ is called the non-principal branch $W_{-1}(x)$. The Lambert W function finds application in statistical mechanics, quantum chemistry, combinatorics, hydrology, iterated exponentiation problems, enzyme kinetics and to solve delay-differential equations [see citations in Corless et al., 1996].

3.2 Peak time of PSP

The system of equations for (2.0.4) with the (2.0.5) can be written in terms of its three components wherein the first two components contribute to current and the third component is the membrane voltage

$$\dot{y}_1 = -\frac{1}{\tau_\alpha} y_1 + \hat{i}\frac{e}{\tau_\alpha} \tag{3.2.1}$$

$$\dot{y}_2 = -\frac{1}{\tau_\alpha} y_2 + y_1 \tag{3.2.2}$$

$$\dot{y}_3 = \frac{1}{C} y_2 - \frac{1}{\tau_m} y_3. \tag{3.2.3}$$

The derivative of (2.0.17) is

$$\dot{u}(t) = \frac{e}{\tau_\alpha}\frac{1}{c}\left[\frac{\frac{-1}{\tau_\alpha}e^{\frac{-t}{\tau_m}} + \frac{1}{\tau_\alpha}e^{\frac{-t}{\tau_\alpha}}}{\left(\frac{1}{\tau_\alpha}-\frac{1}{\tau_m}\right)^2} - \frac{e^{\frac{-t}{\tau_\alpha}} - \frac{1}{\tau_\alpha}te^{\frac{-t}{\tau_\alpha}}}{\left(\frac{1}{\tau_\alpha}-\frac{1}{\tau_m}\right)}\right] \tag{3.2.4}$$

The homogeneous solution y_2 is

$$y_2^h = e^{-\frac{t}{\tau_\alpha}} \tag{3.2.5}$$

and the particular solution that vanishes at $t = 0$ is

$$\begin{aligned} y_2(t) &= y_2^h \int_0^t (y_2^h(t'))^{-1} y(t')\, dt' \\ &= e^{-\frac{t}{\tau_\alpha}} \int_0^t e^{\frac{t'}{\tau_\alpha}} y_1(0) e^{-\frac{t'}{\tau_\alpha}}\, dt' \\ \implies y_2(t) &= y_1(0) t e^{\frac{-t}{\tau_\alpha}}. \end{aligned} \tag{3.2.6}$$

We can determine the maximum of the α-function from $\dot{y}_2 = 0$

$$\begin{aligned} y_1(0)\left(\frac{-1}{\tau_\alpha}\right) t e^{\frac{-t}{\tau_\alpha}} + y_1(0) e^{\frac{-t}{\tau_\alpha}} &= 0 \\ \implies y_1(0)\left[\underbrace{\frac{-t}{\tau_\alpha} + 1}_{(t=\tau_\alpha)\Rightarrow 0}\right] e^{\frac{-t}{\tau_\alpha}} &= 0 \end{aligned} \tag{3.2.7}$$

From (3.2.6) and (3.2.7)

$$y_2(\tau_\alpha) = y_1(0)\, \tau_\alpha e^{-1} \tag{3.2.8}$$

From (2.0.5) and (2.0.10) we have the initial relation for computation of peak time of PSP given by $0 = \dot{y}_3 = \frac{1}{C} y_2 - \frac{1}{\tau_m} y_3$

$$0=\frac{1}{C}\left[y_2(0)e^{\frac{-t}{\tau_\alpha}}+y_1(0)te^{\frac{-t}{\tau_\alpha}}\right]-\frac{1}{\tau_m}\left[y_1(0)\frac{1}{C}\left(\frac{e^{\frac{-t}{\tau_m}}-e^{\frac{-t}{\tau_\alpha}}}{\left(\frac{1}{\tau_\alpha}-\frac{1}{\tau_m}\right)^2}-\frac{te^{\frac{-t}{\tau_\alpha}}}{\left(\frac{1}{\tau_\alpha}-\frac{1}{\tau_m}\right)}\right)+y_2(0)\frac{1}{C}\frac{\left(e^{\frac{-t}{\tau_m}}-e^{\frac{-t}{\tau_\alpha}}\right)}{\left(\frac{1}{\tau_\alpha}-\frac{1}{\tau_m}\right)}+y_3(0)e^{\frac{-t}{\tau_m}}\right] \tag{3.2.9}$$

Re-arranging the expression,

$$\begin{aligned}
0 &= y_1(0)\left[te^{\frac{-t}{\tau_\alpha}}\left(1+\frac{1}{\tau_m\left(\frac{1}{\tau_\alpha}-\frac{1}{\tau_m}\right)}\right)-\frac{1}{\tau_m}\left(\frac{e^{\frac{-t}{\tau_m}}-e^{\frac{-t}{\tau_\alpha}}}{\left(\frac{1}{\tau_\alpha}-\frac{1}{\tau_m}\right)^2}\right)\right]+y_2(0)\left[e^{\frac{-t}{\tau_\alpha}}-\frac{1}{\tau_m}\frac{\left(e^{\frac{-t}{\tau_m}}-e^{\frac{-t}{\tau_\alpha}}\right)}{\left(\frac{1}{\tau_\alpha}-\frac{1}{\tau_m}\right)}\right]-\frac{C}{\tau_m}y_3(0)e^{\frac{-t}{\tau_m}} \\
0 &= y_1(0)\left[\frac{\tau_m}{\tau_m-\tau_\alpha}te^{\frac{-t}{\tau_\alpha}}-\frac{1}{\tau_m}\frac{\left(e^{\frac{-t}{\tau_m}}-e^{\frac{-t}{\tau_\alpha}}\right)}{\left(\frac{1}{\tau_\alpha}-\frac{1}{\tau_m}\right)^2}\right]+y_2(0)\left[e^{\frac{-t}{\tau_\alpha}}-\frac{\left(e^{\frac{-t}{\tau_m}}-e^{\frac{-t}{\tau_\alpha}}\right)}{\tau_m\left(\frac{1}{\tau_m}-\frac{1}{\tau_\alpha}\right)}\right]-\frac{C}{\tau_m}y_3(0)e^{\frac{-t}{\tau_m}}
\end{aligned} \tag{3.2.10}$$

we define

$$\left(\frac{1}{\tau_\alpha}-\frac{1}{\tau_m}\right)=b \quad \text{and} \quad \frac{\tau_m}{\tau_\alpha}=a, \tag{3.2.11}$$

where a is the ratio of the two constants and must be greater than 1. With a and b we rewrite the last expression as

$$0 = y_1(0)\left[\frac{1}{\left(1-\frac{1}{a}\right)}te^{\frac{-t}{\tau_\alpha}}-\frac{1}{\tau_m}\frac{\left(e^{\frac{-t}{\tau_m}}-e^{\frac{-t}{\tau_\alpha}}\right)}{b^2}\right]+y_2(0)\left[e^{\frac{-t}{\tau_\alpha}}-\frac{\left(e^{\frac{-t}{\tau_m}}-e^{\frac{-t}{\tau_\alpha}}\right)}{\tau_m b}\right]-\frac{C}{\tau_m}y_3(0)e^{\frac{-t}{\tau_m}}$$

and get,

$$\begin{aligned}
e^{\frac{-t}{\tau_m}}\left[y_1(0)\frac{1}{b^2}\frac{1}{\tau_m}+y_2(0)\frac{1}{\tau_m b}+\frac{C}{\tau_m}y_3(0)\right] &= e^{\frac{-t}{\tau_\alpha}}\left[y_1(0)\left(\frac{1}{\left(1-\frac{1}{a}\right)}t+\frac{1}{\tau_m b^2}\right)+y_2(0)\left(1+\frac{1}{\tau_m b}\right)\right] \\
\underbrace{e^{\left(\frac{-1}{\tau_m}+\frac{1}{\tau_\alpha}\right)t}}_{b} &= \frac{y_1(0)\left(\frac{t}{\left(1-\frac{1}{a}\right)}+\frac{1}{\tau_m b^2}\right)+y_2(0)\left(1+\frac{1}{\tau_m b}\right)}{\frac{1}{\tau_m b}\left(y_1(0)\frac{1}{b}+y_2(0)\right)+\frac{C}{\tau_m}y_3(0)}.
\end{aligned} \tag{3.2.12}$$

In order to solve for t we bring the expression to a form that can be solved by the Lambert W function

$$e^{bt}\left[\frac{1}{\tau_m b}\left(y_1(0)\frac{1}{b}+y_2(0)\right)+\frac{C}{\tau_m}y_3(0)\right]-\left[y_1(0)\left(\frac{t}{\left(1-\frac{1}{a}\right)}+\frac{1}{\tau_m b^2}\right)+y_2(0)\left(1+\frac{1}{\tau_m b}\right)\right]=0 \tag{3.2.13}$$

or

$$e^{bt} - \left[\frac{y_1(0)\left(\frac{t}{\left(1-\frac{1}{a}\right)} + \frac{1}{\tau_m b^2} \right) + y_2(0)\left(1 + \frac{1}{\tau_m b}\right)}{\frac{1}{\tau_m b}\left(y_1(0)\frac{1}{b} + y_2(0)\right) + \frac{C}{\tau_m} y_3(0)} \right] = 0. \tag{3.2.14}$$

We now define

$$s = bt, \tag{3.2.15}$$

where s is scaled time and must be positive. (3.2.14) is of the form

$$e^s = a's + f \tag{3.2.16}$$

Consider (3.2.11)

$$\begin{aligned} b\left(1 - \frac{1}{a}\right) &= \left(\frac{1}{\tau_\alpha} - \frac{1}{\tau_m}\right)\left(1 - \frac{\tau_\alpha}{\tau_m}\right) \\ &= \frac{\left(\tau_m - \tau_\alpha\right)^2}{\tau_m^2 \tau_\alpha} = \frac{\left(\tau_m - \tau_\alpha\right)^2}{\left(\tau_m \tau_\alpha\right)^2} \tau_\alpha \\ &= \tau_\alpha b^2. \end{aligned} \tag{3.2.17}$$

Writing (3.2.14) explicitly in terms of (3.2.16)

$$a' = \frac{y_1(0)\frac{1}{\tau_\alpha b^2}}{\frac{1}{\tau_m b}(y_1(0)\frac{1}{b} + y_2(0)) + \frac{C}{\tau_m} y_3(0)} \tag{3.2.18}$$

$$f = \frac{y_1(0)\frac{1}{\tau_m b^2} + y_2(0)\left(1 + \frac{1}{\tau_m b}\right)}{\frac{1}{\tau_m b}\left(y_1(0)\frac{1}{b} + y_2(0)\right) + \frac{C}{\tau_m} y_3(0)} \tag{3.2.19}$$

Lambert W solves (3.1.1)

$$\begin{aligned} 1 &= (f + a's)e^{-s} \\ \frac{-1}{a} &= \left(-\frac{f}{a'} - s\right) e^{-s} \\ \underbrace{\frac{-1}{a'} e^{\frac{f}{a'}}}_{} &= \left(\underbrace{\frac{-f}{a'} - s}_{w}\right) e^{\left(\underbrace{\frac{-f}{a'} - s}_{w}\right)} \\ s &= -W\left(\frac{-1}{a'} e^{\frac{-f}{a'}}\right) - \frac{f}{a'}. \end{aligned} \tag{3.2.20}$$

Plugging in (3.2.18) and (3.2.19) in (3.2.20) the peak time of PSP reads

$$s = -W\left(-\frac{\frac{1}{\tau_m b}(y_1(0)\frac{1}{b} + y_2(0)) + \frac{C}{\tau_m}y_3(0)}{y_1(0)\frac{1}{\tau_\alpha b^2}} e^{-\frac{y_1(0)\frac{1}{\tau_m b^2} + y_2(0)\left(1+\frac{1}{\tau_m b}\right)}{y_1(0)\frac{1}{\tau_\alpha b^2}}}\right) - \frac{y_1(0)\frac{1}{\tau_m b^2} + y_2(0)\left(1+\frac{1}{\tau_m b}\right)}{y_1(0)\frac{1}{\tau_\alpha b^2}}. \tag{3.2.21}$$

Preliminary checks for validity and accuracy of this solution was done using Python [Python Software Foundation, 2008] and then implemented in NEST [Diesmann and Gewaltig, 2002].

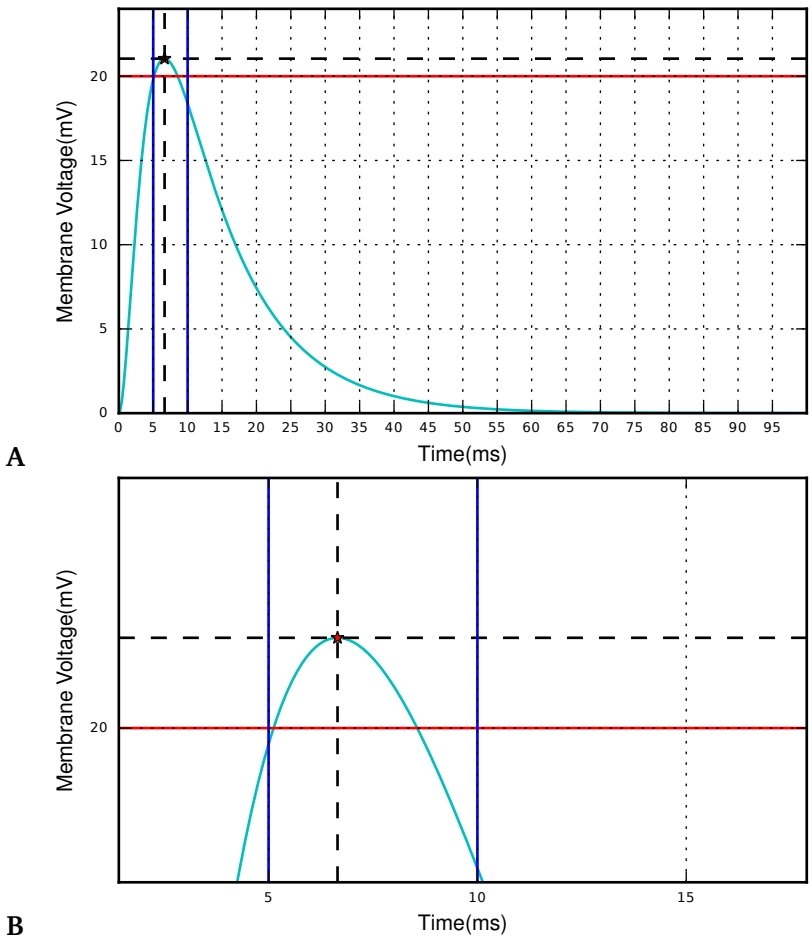


Figure 3.2.1: Lambert W function calculates the peak time of PSP (Python Implementation). Figure shows the results of the implementation of (3.2.21). (**A**) Dynamics of membrane voltage as a function of time. (**B**) Peak time $t_{\max} = 6.650$ ms between two consecutive intervals $t_{\text{left}} = 5$ ms and $t_{\text{right}} = 10$ ms. The dashed vertical line marks the position of the peak predicted by (3.2.21) as $t_{\max} = \frac{s}{b}$. The dashed horizontal line is given by (2.0.10) as $V(t_{\max}) = y_3(t_{\max})$. Threshold voltage $\Theta = 20$ mV. Other parameters are $y_1(0) = 20$, $y_2(0) = 0$, $y_3(0) = 0$ and other standard parameters as in 2.

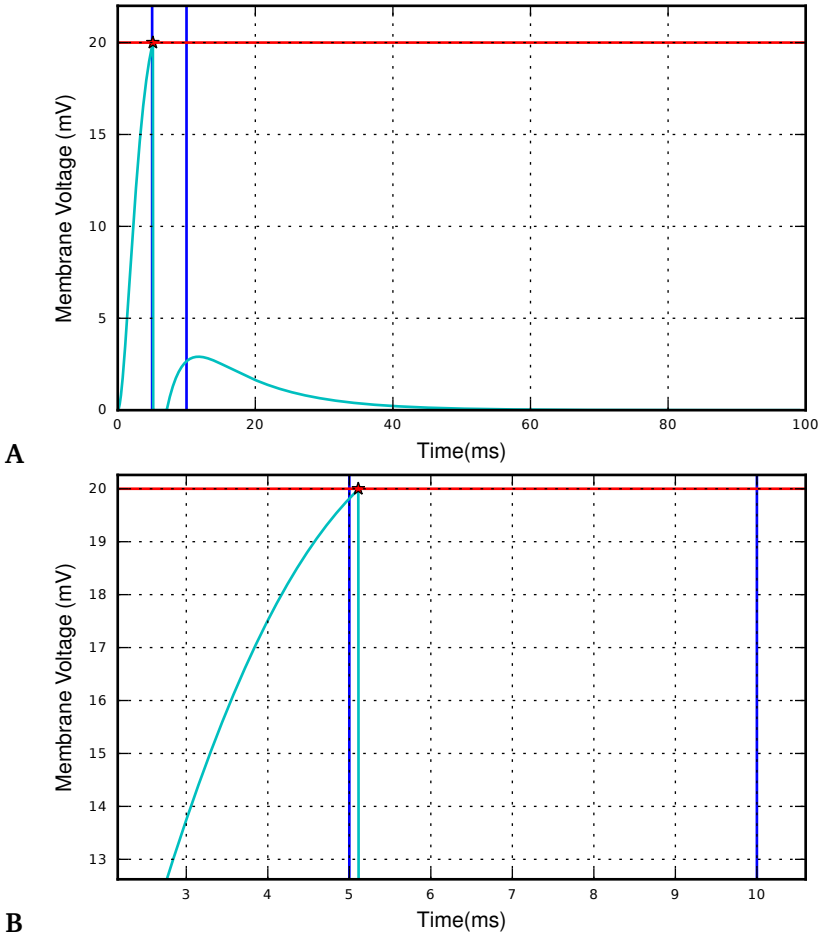


Figure 3.2.2: Lambert W function calculates peak time of PSP (NEST Implementation) for the same conditions as in Figure 3.2.1 on page 25.

The peak time calculated by (3.2.21) coincides with maximum assumed by the time course of the membrane potential obtained from the Python implementation Figure 3.2.1 on page 25. The difference to the NEST implementation Figure 3.2.2 on page 26 is the existence of the threshold $\Theta = 20\,\mathrm{mV}$. After hitting the threshold the voltage is reset to the resting voltage (here at 0 mV). Up to the point of theshold crossing for the same initial conditions, the both implementations agree.

Real-valued solutions for the above equation lie in the non-principal branch of the Lambert W function. This can be answered by extending the reasoning that has already been established in Diesmann (unpubl.).

The argument of the Lambert W solution (3.2.21) reads

$$\begin{aligned}\arg(W) &= -\frac{\frac{1}{\tau_m b}(y_1(0)\frac{1}{b}+y_2(0))+\frac{C}{\tau_m}y_3(0)}{y_1(0)\frac{1}{\tau_\alpha b^2}}\, e^{\underbrace{-\frac{y_1(0)\frac{1}{\tau_m b^2}+y_2(0)\left(1+\frac{1}{\tau_m b}\right)}{y_1(0)\frac{1}{\tau_\alpha b^2}}}_{\mathrm{p}}} \\ &= \frac{-1}{a}\,\frac{\frac{1}{b}\left(y_1(0)\frac{1}{b}+y_2(0)\right)+Cy_3(0)}{y_1(0)\frac{1}{b^2}}\, e^{p}.\end{aligned} \tag{3.2.22}$$

From (3.2.11) and (3.2.21) we have

$$W < \frac{-1}{a} \tag{3.2.23}$$

since $s > 0$ and $a > 1$ in the interval $\frac{-1}{e} < \arg(W) < 0$. This interval is bounded by the line $e\,x$ Figure 3.1.1 on page 20 which leads to the condition that

$$\frac{-1}{a} < \mathrm{W} < \mathrm{e\,x} \tag{3.2.24}$$

that means that there can be no solution for this equation in the principal branch.

3.3 Critical conditions

3.3.1 Initial condition $y_1(0) = 0$

Looking at the solution (3.2.21) we see that for the case $y_1(0) = 0$ there is no solution. This motivates the need for an expression for this special case. Starting from (3.2.12) we obtain

$$\underbrace{e^{\left(\frac{-1}{\tau_m}+\frac{1}{\tau_\alpha}\right)t}}_{b} = \frac{y_1(0)\left(\frac{t}{\left(1-\frac{1}{a}\right)}+\frac{1}{\tau_m b^2}\right)+y_2(0)\left(1+\frac{1}{\tau_m b}\right)}{\frac{1}{\tau_m b}\left(y_1(0)\frac{1}{b}+y_2(0)\right)+\frac{C}{\tau_m}y_3(0)}$$

$$\begin{aligned} e^{bt} &= \frac{y_2(0)\left(1+\frac{1}{\tau_m b}\right)}{\frac{1}{\tau_m b}y_2(0)+\frac{C}{\tau_m}y_3(0)} \\ s &= \log\left(\frac{y_2(0)\left(1+\frac{1}{\tau_m b}\right)}{\frac{1}{\tau_m b}y_2(0)+\frac{C}{\tau_m}y_3(0)}\right) \\ t &= \frac{1}{b}\log\left(\frac{y_2(0)\left(1+\frac{1}{\tau_m b}\right)}{\frac{1}{\tau_m b}y_2(0)+\frac{C}{\tau_m}y_3(0)}\right) \end{aligned} \tag{3.3.1}$$

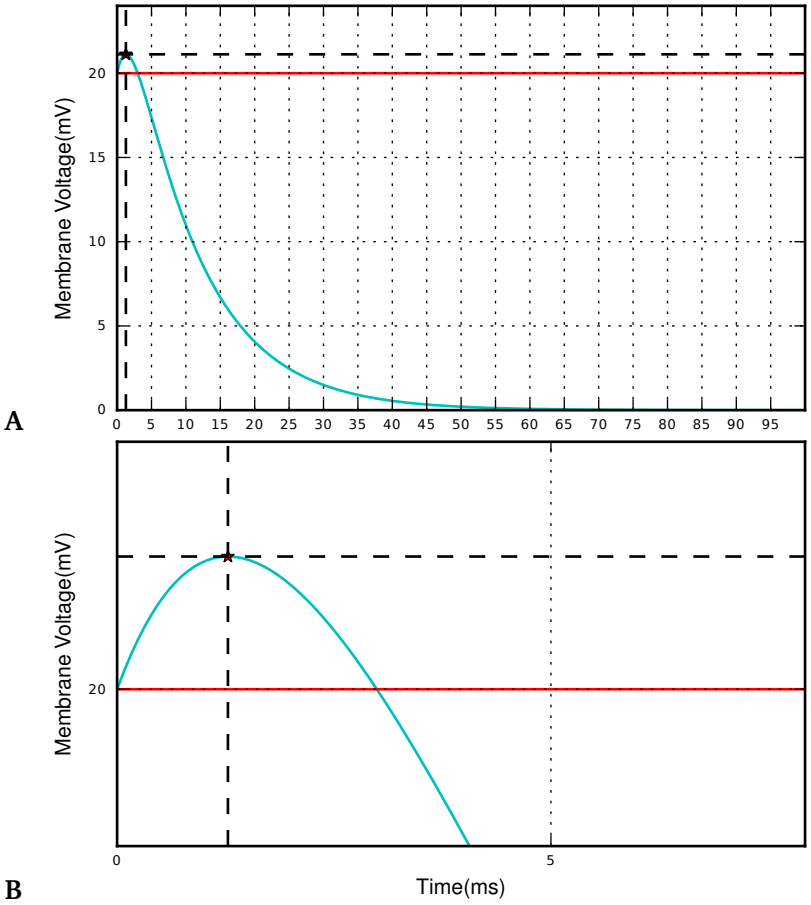


Figure 3.3.1: Plots showing the calculation of peak time for the special case when $y_1(0) = 0$ from (3.3.1).**(A)** The dynamics of membrane voltage as a function of time for initial conditions $y_1(0) = 0$, $y_2(0) = 10$, $y_3(0) = 0.2$ wherein the (3.3.1) computes the peak time. The dashed vertical and horizontal lines mark this peak time and the red line marks the threshold voltage($\Theta = 20$ mV). **(B)** The same figure in a finer resolution where $t_{\max} = 1.277$ ms.

3.3.2 Real-valued solution of the Lambert W function

Considering the computational cost of this function (3.2.21) that is needed to compute the peak time at each instance where a spike miss may occur, it is intuitive to think of an efficient way to eliminate unnecessary computations. It is therefore useful if a certain set of initial conditions could directly be identified as not leading to a maximum between two time intervals. From Corless et al. [1996] we have that a real valued solution seizes to exist (cf. Figure 3.1.1 on page 20) if

$$\arg(\mathrm{W}) < \frac{-1}{\mathrm{e}}. \tag{3.3.2}$$

The idea is to check what range of initial conditions of the membrane voltage y_3 this condition (3.3.2) corresponds to. From (3.2.22) and (3.3.2) we have

$$\frac{-1}{e} > -\frac{\frac{1}{\tau_m b}(y_1(0)\frac{1}{b} + y_2(0)) + \frac{C}{\tau_m}y_3(0)}{y_1(0)\frac{1}{\tau_\alpha b^2}} e^{-\frac{y_1(0)\frac{1}{\tau_m b^2} + y_2(0)\left(1+\frac{1}{\tau_m b}\right)}{y_1(0)\frac{1}{\tau_\alpha b^2}}} \tag{3.3.3}$$

Multiplying both sides of (3.3.2) by -1 and flipping the unequal sign yields after insertion of (3.2.22)

$$\frac{1}{e} > \frac{\frac{1}{\tau_m b}(y_1(0)\frac{1}{b} + y_2(0)) + \frac{C}{\tau_m}y_3(0)}{y_1(0)\frac{1}{\tau_\alpha b^2}} e^{-\frac{y_1(0)\frac{1}{\tau_m b^2} + y_2(0)\left(1+\frac{1}{\tau_m b}\right)}{y_1(0)\frac{1}{\tau_\alpha b^2}}} \tag{3.3.4}$$

Solving for $y_3(0)$ we have

$$y_3(0) < \frac{1}{C}\left(\frac{a}{b^2}y_1(0)e^{\frac{y_1(0)\frac{1}{\tau_m b^2} + y_2(0)\left(1+\frac{1}{\tau_m b}\right)}{y_1(0)\frac{1}{\tau_\alpha b^2}} - 1} - \frac{1}{b}(y_1(0)\frac{1}{b} + y_2(0))\right) \tag{3.3.5}$$

The exponent can be re-written as

$$\begin{aligned}
\frac{y_1(0)\frac{1}{\tau_m b^2} + y_2(0)\left(1+\frac{1}{\tau_m b}\right)}{y_1(0)\frac{1}{\tau_\alpha b^2}} - 1 &= \frac{y_1(0)\frac{1}{\tau_m b^2} + y_2(0)\left(1+\frac{1}{\tau_m b}\right) - y_1(0)\frac{1}{\tau_\alpha b^2}}{y_1(0)\frac{1}{\tau_\alpha b^2}} \\
&= \frac{y_2(0)\left(1+\frac{1}{\tau_m b}\right) - \frac{y_1(0)}{b}}{y_1(0)\frac{1}{\tau_\alpha b^2}} \\
&= \frac{y_2(0)}{y_1(0)}\left(1+\frac{1}{\tau_m b}\right)\tau_\alpha b^2 - \tau_\alpha b \\
&= b\frac{y_2(0)}{y_1(0)}\left(\tau_\alpha b + \frac{1}{a}\right) - \tau_\alpha b \\
&= b\frac{y_2(0)}{y_1(0)} - \tau_\alpha b
\end{aligned} \tag{3.3.6}$$

From (3.3.5) and (3.3.6) we get

$$y_3(0) < \frac{1}{C}\left(\frac{a}{b^2}y_1(0)e^{\left(b\frac{y_2(0)}{y_1(0)} - \tau_\alpha b\right)} - \frac{1}{b}(y_1(0)\frac{1}{b} + y_2(0))\right) \tag{3.3.7}$$

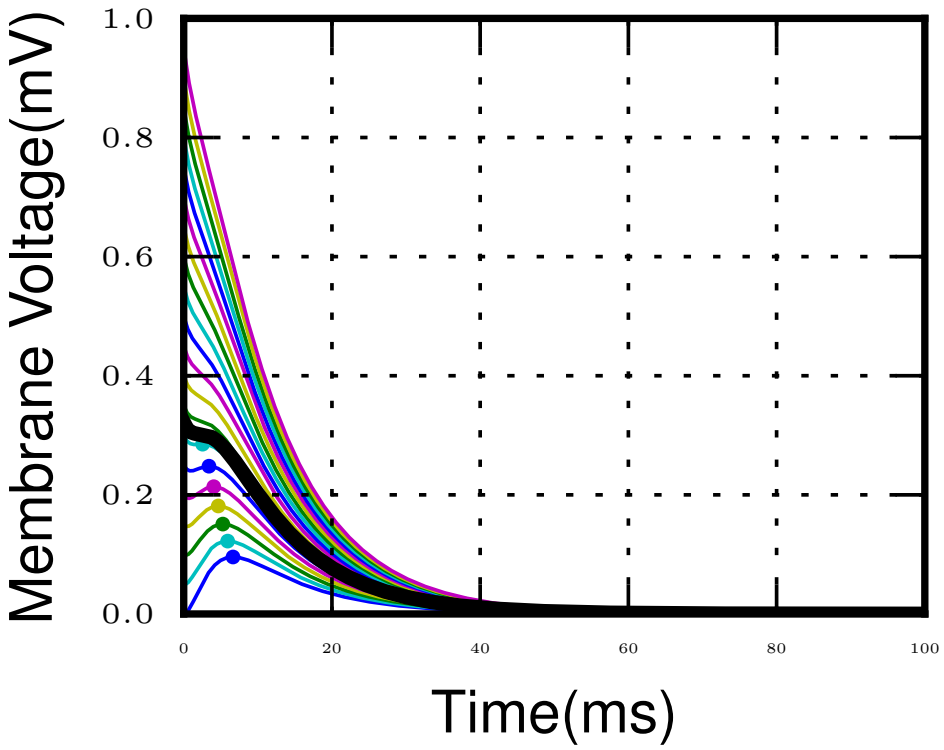


Figure 3.3.2: Dynamics of membrane voltage as a function of time for iteratively increasing value of $y_3(0)$ which is bound by the condition (3.3.5). The black line indicates the critical point, given by (3.3.5), above which the membrane voltage cannot have a peak time that could cause a spike excursion.

This upper limit for the initial condition of membrane voltage is included as a preliminary check before the spike test is actually carried out in the lossless model. Any initial condition above this value for sure cannot lead to a threshold crossing. This is illustrated in Figure 3.3.2 on page 30.

4 Continuous-time spiking neuron models

4.1 NEST

The Neural Simulation Technology (NEST) initiative is a collaborative effort to develop an open simulation framework for biologically realistic neuronal networks. A NEST simulation tries to mimic an electrophysiological experiment, which includes two conceptual domains: the description of the neuronal system and the description of the experimental setup and protocol. NEST is written in object-oriented style (C++) [Diesmann and Gewaltig, 2002]. The first user-interface to NEST [Gewaltig and Diesmann, 2007] was the simulation language SLI, a stack-based language derived from PostScript. A more conveniently usable interface to NEST is the PyNEST, the Python interface to NEST. NEST defines the neural world in terms of directed, weighted graph of nodes and connections. Nodes are neurons, connections are characterized by a configurable fixed delay, and a weight that can be static or dynamic. Several models for nodes and connections are built into NEST. Their parameters are accessed via dictionaries [Eppler et al., 2009].

4.2 $\alpha-$ neuron model

In the previous chapter Chapter 2 an outline of the canonical neuron model has been discussed. The idea of combining exact integration of sub-threshold membrane dynamics with interpolation to calculate the precise spike timing has been motivated in Chapter 2 using illustrations Figure 2.0.2 on page 14 and Figure 2.0.3 on page 15. This chapter discusses an overview of the general algorithmic structure of the alpha models: `canonical` and `lossless`.

The key function that handles spike events in the alpha model is the `update` function. The update and spike handling function applies the time evolution operator and calls `the emit_spike, propagate` function and other auxiliary functions that processes events in the time grid. The `update` function moves the state of the neuron from t_0 to $t_0 + dt$ either in steps of resolution h, if there are no incoming spike events in the time interval or from event to event, as retrieved from the spike queue. The dynamics from one point to the other is calculated using the propagator matrix. For steps in which there are no incoming spike events, it is handled by the fixed pre-computed propagator matrix.

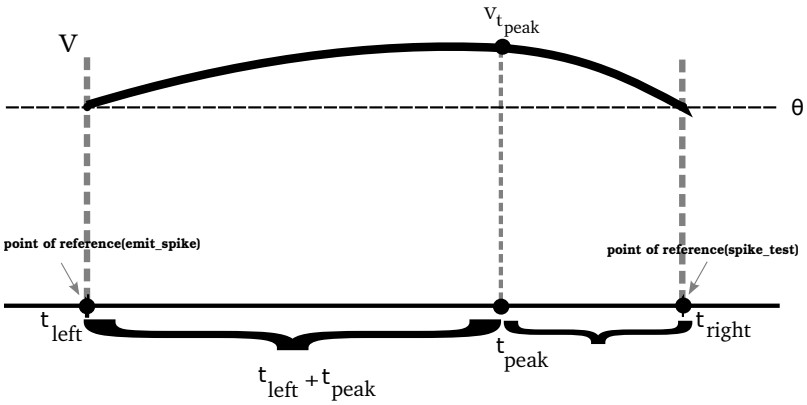


Figure 4.2.1: Time grid with the position of maximum membrane potential at t_{peak}. The points of reference for the functions `emit_spike` and `spike_test` functions are different: The `emit_spike` function measures time backward i.e. t_{left} is the point of reference. The `spike_test` function operates with t_{right} as its reference point.

`emit_spike` Algorithm 2 on page 37 emits a single spike and resets the neuron. The function assumes that the the membrane potential of the neuron was below threshold at the beginning of a time step and above afterwards. Further `emit_spike` calls `thresh_find` that determines the time of threshold crossing using interpolation (linear, cubic or quadratic) [Morrison et al., 2007b] which has been briefly discussed in Chapter 2. `spike_test` Algorithm 4 on page 38 (part of the `lossless` neuron model) in addition to the interpolation function captures spike misses between points on the time grid. The point of reference of computation in the `emit_spike` function is the beginning of the interval and that of the `spike_test` function is the end of the interval, as illustrated in Figure 4.2.1 on page 32.

The `update` function Algorithm 1 on page 33 initially checks for realistic conditions of the time grid and if the membrane potential was initially set above threshold it calls the `emit_instant_spike` function that resets the membrane voltage and switches the neuron to refractoriness. To make the neuron return from refractoriness after the refractory time, it

places a pseudoevent into the ring buffer, that, when processed, ends the refractory period.

```
1  method update_neuron(origin, t_left, t_right)
2    assert(t_right ≥ 0 and t_left < t_right)
3    if(t_left==0)
4      prepare for delivery of input spike
5    if(neuron is suprathreshold initially)
6      emit_instant_spike(origin, t_left, h)
7    for each time step within range (t_left, t_right)
8      if(neuron returns from refractoriness)
9        place pseudoevent to mark the end
10     memorize state variables before spike-time interpolation
11     //spike_test_case_1: no incoming events
12     if(no incoming spikes)
13       if(not refractory)
14         y ← P(h) y
15         if(membrane potential ≥ threshold)
16           emit_spike // Alg. (2)
17         else
18           spike_test_wrapper // Alg. (5)
19     //spike_test_case_2: incoming events
20     else
21       last_offset ← h
22       do
23       // event offsets are measured backward
24       // from the right end of the interval
25         ministep ← last_offset − event_offset
26         y ← P(ministep) y // Eq. (2.0.10)
27         if(membrane potential ≥ threshold)
28           emit_spike
29         else
30           spike_test_wrapper
31         if(end_of_refract)
32           is_refractory ← false
33         else
34           apply received spike input to change initial conditions
35           memorize state variables before interpolation
36         last_offset ← event_offset
37       while(there are incoming spikes, set event_offset)
38     //spike_test_case_3: no events remaining, do remainder
39       if(last_offset > 0)
40
41         y ← P(last_offset) y
42       if(membrane potential ≥ threshold)
43         emit_spike
44       else
45         spike_test_wrapper
46     set new input current
47     record data
48   endfor
```

Algorithm 1: update function algorithm: The update function acts as a time-evolution operator and propagates the state of the neuron in time from t to $t+h$.

This is followed by the principal part of the `update` function where the dynamics of the neuron is propagated by updating membrane voltage at every checkpoint by applying the pre-computed fix propagator matrix (where there are no incoming events) and propagating the dynamics using (2.0.10) where there are incoming events. Whenever there is a threshold crossing, the `emit_spike` function `Algorithm` 2 on page 37 a spike event is registered and the membrane voltage of the neuron is reset to the resting voltage after which the neuron enters a short period of refractoriness. This `emit_spike` function Algorithm 2 is integrated into the `spike_test` `Algorithm` 4 on page 38 function in case of the `lossless` model. The flow of the `update` function `Algorithm` 1 on page 33 is distinguished as three cases based on arrival (or non-arrival) of events and discretization of the time interval under consideration.

A

	case 1	case 2	case 3
`origin`	`origin`	`origin`	`origin`
`lag`	`lag`	`lag`	`lag`
t_0	0	$h-$`last_offset`	$h-$`last_offset`
dt	h	`ministep`	`last_offset`

B

	case 1	case 2	case 3
t_0	T	T	T
t_{left}	h	`ministep`	`last_offset`
t_{max}	t_{max}	t_{max}	t_{max}

Table 4.2.1: Delineation of the different conditions `that calls for (A)` `emit_spike_(&origin, lag,` t_0`,` dt`)` `Algorithm` 2 on page 37 and **(B)** spike_test_(t_0, t_{left}, &t_{max}) `Algorithm` 4 on page 38 functions that are called in the `update` function `Algorithm` 1 on page 33 . The columns indicate different spike test cases as illustrated in Figure 4.2.2 on page 35, Figure 4.2.3 on page 36 and Figure 4.2.4 on page 37. The rows indicate the arguments taken by the function in the respective cases. The parameter `origin` refers to the timestamp at the beginning of the time slice, `lag` is the timestep within each slice, t_0 is the beginning of each mini-timestep and dt is the duration of the mini-timestep. $T = \text{origin} + \text{lag}$ is the time at the start of each update step.

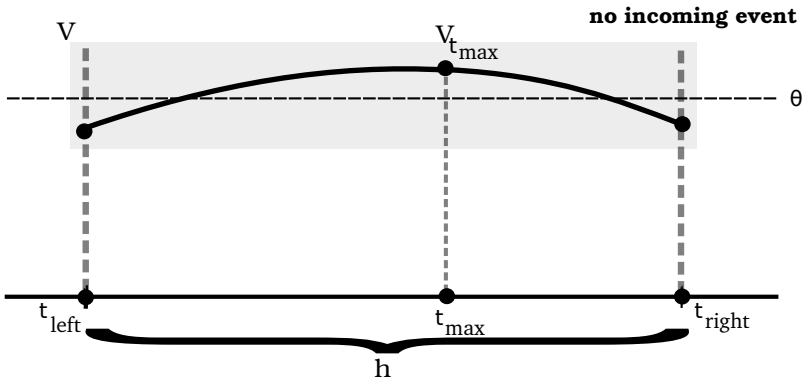


Figure 4.2.2: Illustration of the `spike_test_case_1` referring to line 11 of `Algorithm` 1 on Page 33 where there are no incoming spike events and the dynamics is propagated in intervals of h (indicated by the gray box). The membrane potential is lower than the threshold at t_{left} and t_{right} and the excursion occurs between these two points. Filled black dots show that the values of state variables are known at those points in time.

In `spike_test_case_1` shown in Figure 4.2.2 on page 35 the membrane dynamics is propagated using a fixed propagator matrix since there are no incoming events. The fixed propagator matrix is pre-computed and hence handling this case separately improves performance. If the membrane voltage is above threshold and is detected right away, `the emit_spike` function `Algorithm` 2 on page 37 is called else (in the case of the `lossless` model) the `spike test wrapper Algorithm` 5 on page 39 is called to check whether there has been a spike miss `Algorithm` 4 on page 38 between the two consecutive intervals of the time grid under reference.

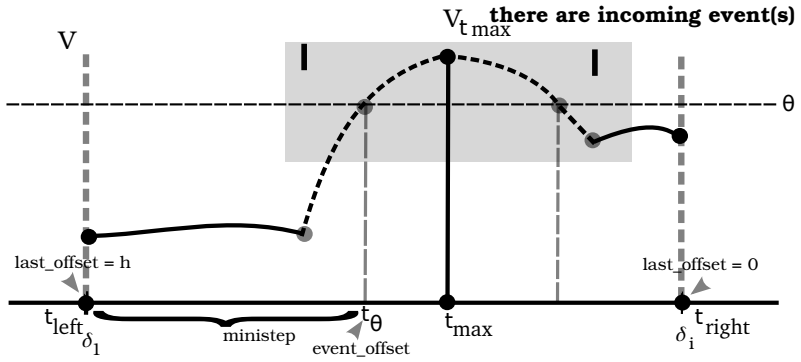


Figure 4.2.3: Illustration of `spike_test_case_2` referring to line 20 of Algorithm 1 where there are incoming events. In this case the interval is divided into offsets and the time is measured backwards Figure 4.2.1 on page 32. A `ministep`, defined in terms of these offsets is propagated (by the `propagate` function) and the check for threshold crossings is carried out in these smaller intervals (indicated by the gray box). If the membrane potential is suprathreshold in these ministeps, `emit_spike` is called, else a `spike_test` is carried out to determine missed excursions. Filled black dots show that the values of state variables are known at those points in time. Gray dots indicate points where the values of state variables are unknown.

In `spike_test_case_2` time within the reference interval is propagated in steps corresponding to the offsets $\delta_1 \leq \delta_2 \leq \ldots \leq \delta_i$ of the incoming spike events. The offsets are measured backward in time, i.e. an offset is h at t_{left} and 0 at t_{right}, as illustrated in Figure 4.2.3 on page 36. The time propagation therefore happens in ministeps that are defined as the difference between the `last offset` last_offset $\equiv \delta_{i-1}$ and the current offset event_offset $\equiv \delta_i$ as ministep = last_offset − event_offset. The dynamics is driven forward by applying the propagator in ministeps Algorithm 3`Algorithm` 3 on page 38 until the end of the interval. Due to an incoming event between t_{left} and t_{right} a crossing of the membrane voltage occurs in one of these ministeps. In the case of the `lossless` model, if the membrane potential is below threshold at the boundaries of the `ministep` the `spike testwrapper Algorithm` 5 on page 39 is called to check for possibility of missed excursions `Algorithm` 4 on page 38 .

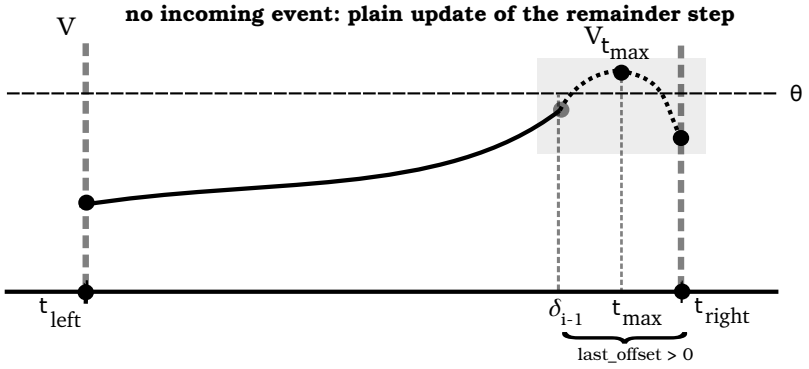


Figure 4.2.4: Illustration of the `spike_test_case_3` referring to line 37 of Algorithm 1 where spiking occurs when `last_offset`> 0 (interval indicated by gray box). Filled black dots show that the values of state variables are known at those points in time. Gray dots indicate points where the values of state variables are unknown.

In `spike_test_case_3`, here illustrated in Figure 4.2.4 on page 37, there are no incoming spikes, but as an after-effect of an incoming event the `last_offset` $\delta_i > 0$. This requires a plain update across the remainder of the timestep using the `propagate` function. If the membrane potential is less than threshold at δ_{i-1} and at t_{right}, in the case of the `lossless` model, the `spike_test` `Algorithm` 4 on page 38 is carried out to check whether there has been a missed spike. If an excursion is detected between these two points, the `emit_spike` function `Algorithm` 2 on page 37 is called. The specifics of these two functions (`emit_spike` and `spike_test`) are highlighted in Table 4.2.1 on page 34.

1 **method** `emit_spike(&origin,lag,`t_0`,`dt`)`
2 spike_offset $= h - (t_0 +$ thresh_find$(dt))$
3 last_spike_offset $\leftarrow$ spike_offset
4 $y_3 \leftarrow U_{\text{reset}}$
5 `is_refractory ← true`
6 `send spike`
7 `return`

Algorithm 2: `emit_spike` function algorithm.

The `emit_spike` function views the timestep with the beginning of interval as its reference as illustrated in Figure 4.2.1 on page 32 and updates `last_offset` to the precise `spike_offset` based on interpolation using the `thresh_find` function. It resets the membrane potential to resting membrane potential sets the neuron to refractoriness. Membrane

voltage remains unchanged during refractory period of neuron.

```
1  method propagate(dt)
2    if(not refractory)
3      compute propagator matrix
4      y ← P(dt)y
```

Algorithm 3: `Propagate` function algorithm Eq.2.0.10.

The propagate function moves the state of neuron by applying the propagator matrix (2.0.8) to every timestep i.e. in steps of dt.

```
1  boolean, double spike_test(t_left, t_right)
2    if(y_3(0) violates critical condition) // Eq. (3.3.7)
3      return false, 0
4    else
5      if(y_1(0) == 0)
6        compute t_peak special condition // Eq. (3.3.1)
7      else
8        compute t_peak Lambert W function // Eq. (3.2.21)
9        if(t_peak < −t_left && t_peak < 0)
10         propagate(t_peak)// Eq. (2.0.10)
11           t_max = t_left + t_peak
12           if(membrane potential at t_max ≥ threshold)
13
14             return true, t_max //spike miss
15     return false, 0
```

Algorithm 4: `spike_test` function algorithm.

`spike_test` checks for spikes that are otherwise missed. It initially checks for certain critical conditions that has been analytically derived in Section 3.3 which would render running through this test unnecessary thereby optimizing performance. In this case, the function returns `false`, indicating that there are no spikes missed. The peak time is calculated using the Lambert W function [Corless et al., 1996] but for special initial conditions a different expression is used as discussed in Section 3.2. The `spike_test` function views the time grid with the end of the time interval as its reference Figure 4.2.1 on page 32. The computed peak time is checked whether it is located between the two consecutive points on the temporal grid. If this is satisfied and if the membrane potential at this point exceeds the

threshold, then a spike has been certainly missed, and the function returns `true`.

```
1 method spike_test_wrapper
2   store state variables before spike test
3   boolean test_cond = spike_test(timestamp,h,&t_max)
4   if(test_cond)
5     emit_spike(origin,lag,t_0,dt)
6   else
7     update membrane potential
8   memorize the values of synaptic current after spike test
```

Algorithm 5: Spike test wrapper.

The `spike test wrapper` is called in the different instances in which spiking can happen in the `update` function. This executes `spike_test` `Algorithm` 4 on page 38, and emits spike `Algorithm` 2 on page37 to compensate for the missed spike in case there has been a spike miss. In this case, the state of membrane potential is preserved i.e. it holds the peak value of membrane potential. However, if there has been no spike miss, then the values of the state variables are restored to that of the end of the time interval of reference. This `spike_test` implemented in the `lossless` model and complements the interpolation function `thresh_find` existing in the `canonical` model.

4.3 Spike test

In Section 4.2 the different methods in which spike events are handled by the `update` function were discussed. This also corresponds to the different situations Figure 4.2.2 on page 35, Figure 4.2.3 on page 36 and Figure 4.2.4 on page 37 in which spikes may be missed. The idea of the `lossless` model is to detect these spike misses using the `spike_test` that computes peak time using the Lambert W function [Corless et al., 1996] and to call `emit_spike` for every lost spike. To check for the credibility of this test, certain initial conditions were constructed such that a situation where spike miss is certain (here referred to as explicit spike tests). Applying these conditions to the implementation developed for the NEST simulator Gewaltig and Diesmann [2007], we confirmed that the spike test captures these misses.

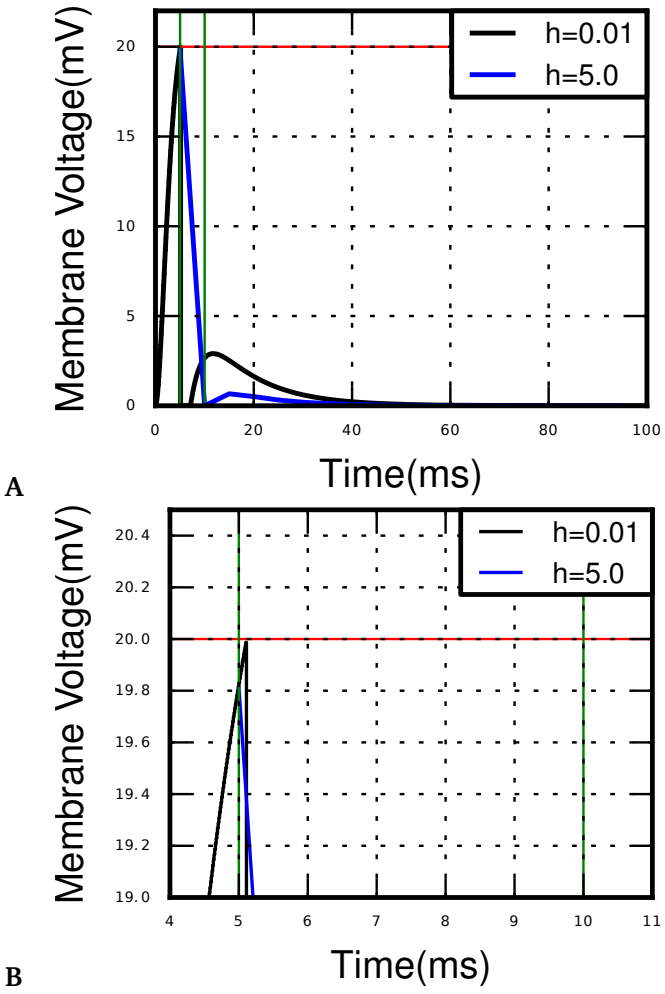


Figure 4.3.1: **spike_test_case_1** of the `Algorithm` 1 on page 33: there are no incoming events: The threshold $\Theta = 20$ mV marked by the red line and the black line shows the membrane potential dynamics for a fine grid resolution ($h = 0.01$ ms) and the blue line for a coarse grid resolution ($h = 5.0$ ms). Panel **(A)** shows that for larger grid resolution an excursion is missed at the beginning of the simulation in a step h marked by the green lines for the initial condition ($y_1(0) = 22$ mV, $y_2(0) = y_3(0) = 0$). Panel **(B)** shows a closer look at the interval of reference ($t_{\text{left}} = 5.0$ ms and $t_{\text{right}} = 10.0$ ms) wherein the threshold crossing is missed for a larger grid size. This can be corrected by including the `spike_test Algorithm` 4 on Page 38 that calculates the precise spike timing. A sketch of fine grid resolution is shown along to compare with an accurately propagated dynamics of membrane potential.

The `spike_test_case_1 Algorithm` 4 on page 38 is the situation where there are no incoming events. In such a condition, a spike is missed between t_{left} and t_{right}. Figure 4.3.1

on page 40 shows that for large grid resolution such excursions are missed but can be caught by the `spike_test Algorithm` 4 on page 38 .

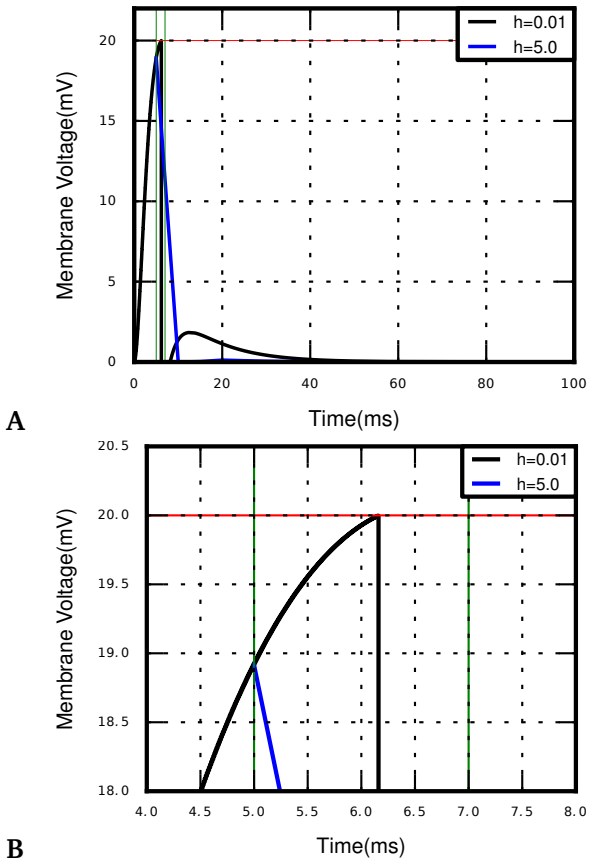


Figure 4.3.2: `spike_test_case_2` of the `Algorithm` 1 on Page 33 . The neuron is connected to a spike generator and there are incoming events: The threshold $\Theta = 20$ mV marked by the red line and the black line shows the membrane potential dynamics for a fine grid resolution ($h = 0.01$ ms) and the blue line for a coarse grid resolution ($h = 5.0$ ms). Panel **(A)** shows that an incoming event causes a threshold crossing in a `ministep` (here between 5.0 ms and 7.0 ms indicated by the green lines) for initial conditions ($y_1(0) = 21$ mV, $y_2(0) = y_3(0) = 0$). Panel **(B)** shows a closer look at the interval of reference ($t_{\text{left}} = 5.0$ ms and $t_{\text{right}} = 10.0$ ms) wherein the threshold crossing is missed for a larger grid size. This can be corrected by including the `spike_test Algorithm` 4 on Page 38 that calculates the precise spike timing. A sketch of fine grid resolution is shown along to compare with an accurately propagated dynamics of membrane potential.

The **spike_test_case_2** Figure 4.2.3 on page 36 is the situation in which there are incoming events. Therefore, spike misses occur most often in this condition. In such a condition,

a spike is missed in a ministep. Figure 4.3.2 on page 41 shows that for large grid resolution such excursions are missed but can be caught by the `spike_test Algorithm` 4 on page 38 .

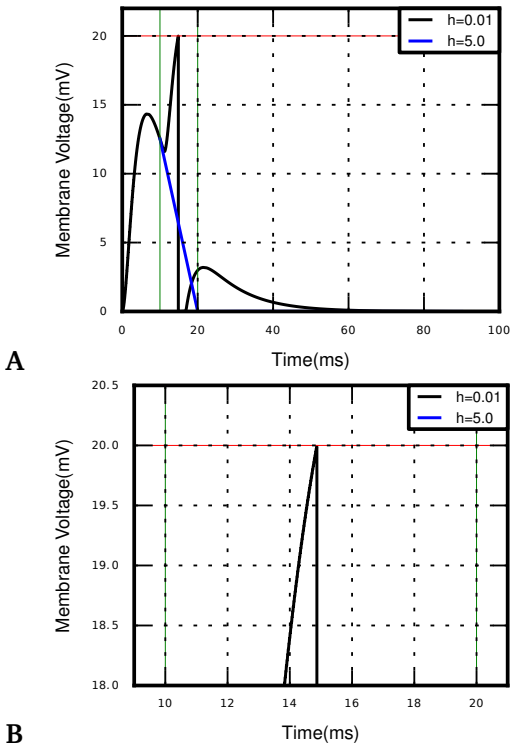


Figure 4.3.3: `spike_test_case_3` of the `Algorithm` 1 on Page 33 : propagation through the remainder of the interval (`last_offset`): The threshold $\Theta = 20$ mV marked by the red line and the black line shows the membrane potential dynamics for a fine grid resolution ($h = 0.01$ ms) and the blue line for a coarse grid resolution ($h = 10.0$ ms). Panel **(A)** shows that an incoming event coming in at the beginning of the first interval that did not lead to threshold crossing for initial conditions $y_1(0) = 15$ mV, $y_2(0) = 10$ mV, $y_3(0) = 0.05$ mV leading to a spike excursion in the second interval between $t_{\text{left}} = 10.0$ ms and $t_{\text{right}} = 20.0$ms (here indicated by the green lines) in the `last_offset` which goes undetected for larger grid resolution as observed in panel **(B)**. This can be corrected by including the `spike_test Algorithm` 4 on page 38 that calculates the precise spike timing. A sketch of fine grid resolution is shown along to compare with an accurately propagated dynamics of membrane potential.

The **spike_test_case_3**Figure 4.2.4 on page 37 is the situation in which an incoming event causes a spike excursion such that `last_offset`> 0. Spikes misses in last offsets can occur for very large grid resolution which can be captured by including the `spike_test` function `Algorithm` 4 on page 38 .

As it can be seen in the above illustrations the inclusion of `spike_test` becomes significant only after a certain resolution. For fine grid, using the `lossless` model does not come with an advantage. This minimum resolution can be intuitively calculated as the width of the excursion i.e. the duration in milliseconds taken for the rise and fall of voltage to occur. Typically in such situations the membrane voltage is below threshold at the boundaries of the interval and spiking occurs in between as in Figure 4.3.4 on page 43.

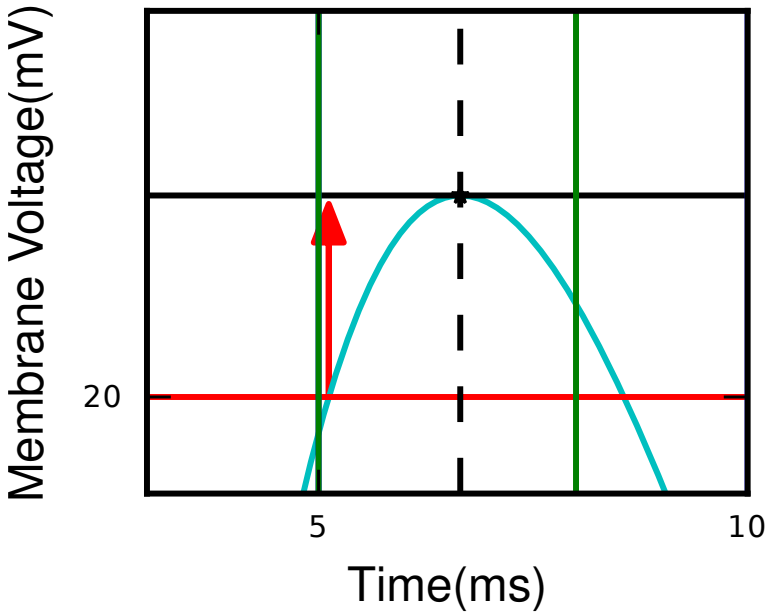


Figure 4.3.4: The minimum grid resolution at which using the `lossless` model is significant is the width of threshold crossing that can be ballparked at ~ 3.2 ms. The threshold is set at $\Theta = 20$ mV and indicated here by the red line. The threshold crossing t_Θ is denoted by the red arrow and the dotted line is drawn to mark the peak time t_{peak}. This result was inferred from `spike_test_case_2 as in Figure 4.2.3 on page 36` .The green lines here indicate the `ministep` in the interval of reference.

With this minimum resolution as reference the simulations are run to test the working and efficiency of the `lossless` model are performed for grid resolution $h = 5.0$ ms, referred to as the standard grid size.

4.4 Neuron embedded in a quasi-network setup

In the previous section it has been verified that the `lossless` α-model catches missed threshold crossings. To test the new α-model in a fluctuating membrane voltage setup it is embedded in a quasi-network setting where it receives excitatory and inhibitory inputs from two poisson generators. This would also help determine the reduction in performance of

the `lossless` model due to the additional tests.

The parameters that characterize the input to the neuron are the mean μ and the variance σ^2 of the total incoming synaptic current. These in turn depend on the excitatory and inhibitory synaptic amplitudes J_e and J_i, excitatory and inhibitory firing rates r_e and r_i of the Poisson sources and the membrane time constant τ_m.

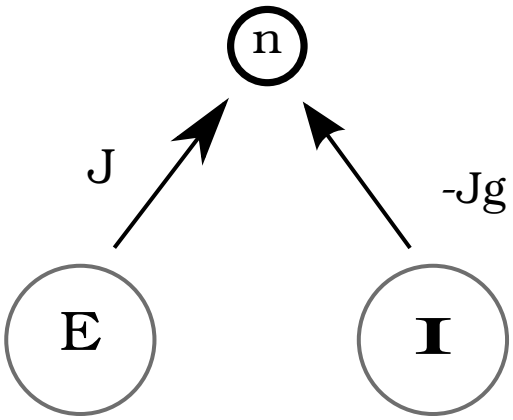


Figure 4.4.1: Excitatory E and Inhibitory I Poisson generators connected to Neuron n. The neuron receives an excitatory synaptic input J and inhibitory synaptic input $-J\,g$.

We will therefore define the overall mean voltage and variance of the neuron's membrane potentialand first check if the simulation results agree to the desired values. The expression for mean and variance for a neuron with $\delta-$shaped post-synaptic currents is given as [Helias et al., 2013]

$$\mu = \tau_m \left(J_e\, r_e + J_i\, r_i\right) \tag{4.4.1}$$

$$\sigma^2 = \frac{\tau_m}{2}\left(J_e^2\, r_e + J_i^2\, r_i\right) \tag{4.4.2}$$

Given μ and σ^2, the expressions for the excitatory and inhibitory firing rate can be analytically derived from above (4.4.1) and (4.4.2) and we arrive at

$$r_e = \frac{2 J_e\, \mu\, \sigma^2}{\tau_m J_i \left(J_e - J_i\right)} \tag{4.4.3}$$

$$r_i = \frac{2\,\sigma^2 - \mu\, J_i}{\tau_m J_e \left(J_e - J_i\right)}. \tag{4.4.4}$$

Since the `lossless` neuron model receives $\alpha-$shaped post-synaptic currents we need to

modify (4.4.1) and (4.4.2) to take the shape of the PSC into account. We are here only interested in an approximate expression valid for short $\tau_\alpha \ll \tau_m$. Therefore we try to approximate the α-shaped pulses by δ-inputs and derive an expression for the new scaled synaptic amplitudes. These will be referred to as J_{exc} for excitatory input and J_{inh} for the inhibitory input.

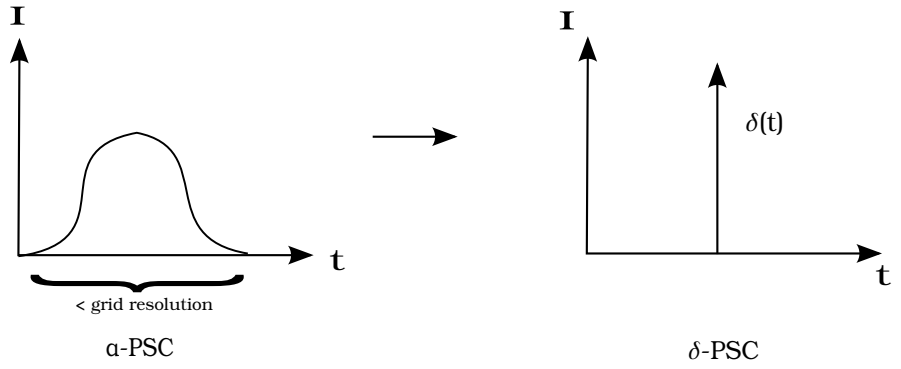


Figure 4.4.2: Approximation of $\alpha-$shaped post-synaptic currents by δ-currents. The width of the α-pulse is less than the grid size and is therefore a fairly good approximation when the dynamics is propagated from one time point in the grid to the next.

We are integrating the PSCs over time to determine their area which we will then choose identical to the factor in front of the δ-current. From (2.0.3) and (2.0.8) this integral can be written as

$$\int_0^\infty y_2(t)\,dt = \int_0^\infty \left(y_1(0)\,te^{-\frac{t}{\tau_\alpha}} + y_2(0)e^{-\frac{t}{\tau_\alpha}} \right) dt, \tag{4.4.5}$$

which, due to linearity, can be re-written as

$$\int_0^\infty y_2(t)\,dt = y_1(0)\int_0^\infty te^{-\frac{t}{\tau_\alpha}}\,dt + y_2(0)\int_0^\infty e^{-\frac{t}{\tau_\alpha}}\,dt. \tag{4.4.6}$$

We define

$$I_1 = y_1(0)\int_0^\infty te^{-\frac{t}{\tau_\alpha}}\,dt \tag{4.4.7}$$

$$I_2 = y_2(0)\int_0^\infty e^{-\frac{t}{\tau_\alpha}}\,dt. \tag{4.4.8}$$

Since we are interested in the effect of a single incoming synaptic event, we use the initial condition $y_2(0) = 0$, so

$$I_2 = 0$$

and applying integration by parts we get

$$\begin{aligned} I_1 &= y_1(0)\left(\underbrace{\left[-t\tau_\alpha e^{-\frac{t}{\tau_\alpha}}\right]_0^\infty}_{=0} + \int_0^\infty \tau_\alpha e^{-\frac{t}{\tau_\alpha}}\,dt\right) \\ &= y_1(0)\left(0 - \left[-\tau_\alpha^2 e^{\frac{-t}{\tau_\alpha}}\right]_0^\infty\right) \\ I_1 &= y_1(0)\tau_\alpha^2. \end{aligned}$$

The solution to (4.4.5) is therefore

$$\int_0^\infty y_2(t)\,dt = y_1(0)\tau_\alpha^2. \tag{4.4.9}$$

The solution of the first component y_1 of the subsystem in (2.0.4) generating the alpha current obeys the differential equation $\dot{y}_1 = -\frac{1}{\tau_\alpha}y_1$ with the initial condition $y_1(0) = \hat{\imath}\frac{e}{\tau_\alpha}$. Hence its solution is

$$y_1(t) = \hat{\imath}\frac{e}{\tau_\alpha}e^{-\frac{t}{\tau_\alpha}} \tag{4.4.10}$$

therefore the synaptic amplitude J (affecting the membrane potential and hence including the factor C in (2.0.5)) can be written as

$$J = C\,y_1(0)\frac{\tau_\alpha}{e} \tag{4.4.11}$$

which gives

$$y_1(0) = \frac{1}{C}\frac{e}{\tau_\alpha}J. \tag{4.4.12}$$

From (4.4.9) and (4.4.12) we get

$$\begin{aligned} \int_0^\infty y_2(t)\,dt &= \frac{eJ\,\tau_\alpha^2}{C\,\tau_\alpha} \\ &= \frac{\tau_\alpha\, eJ}{C} \end{aligned} \tag{4.4.13}$$

where $\tau_\alpha e$ can be thought of as the charge per unit spike. For Poisson processes, the membrane voltage $y_3(t)$ can be given as the convolution of the spike train, a sum of all δ-currents located at the point in time of arrival of the spike, arriving with rate r

$$s(t) = \sum \delta(t - t_k) \tag{4.4.14}$$

and kernel

$$h(t) = e^{-\frac{t}{\tau_m}} \tag{4.4.15}$$

as

$$y_3(t) = (s * h)(t). \tag{4.4.16}$$

According to Campbell's theorem [Campbell, 1909], the mean voltage μ is

$$\begin{aligned} \mu &= \langle y_3(t) \rangle \\ &= \lim_{T\to\infty} \frac{1}{T} \int_0^T y_3(t)\, dt \\ &= \lim_{T\to\infty} \frac{1}{T} \int_0^T (s * h)(t)\, dt \\ &\overset{\text{Campbell}}{=} r \int_0^\infty e^{-\frac{t}{\tau_m}}\, dt \end{aligned} \tag{4.4.17}$$

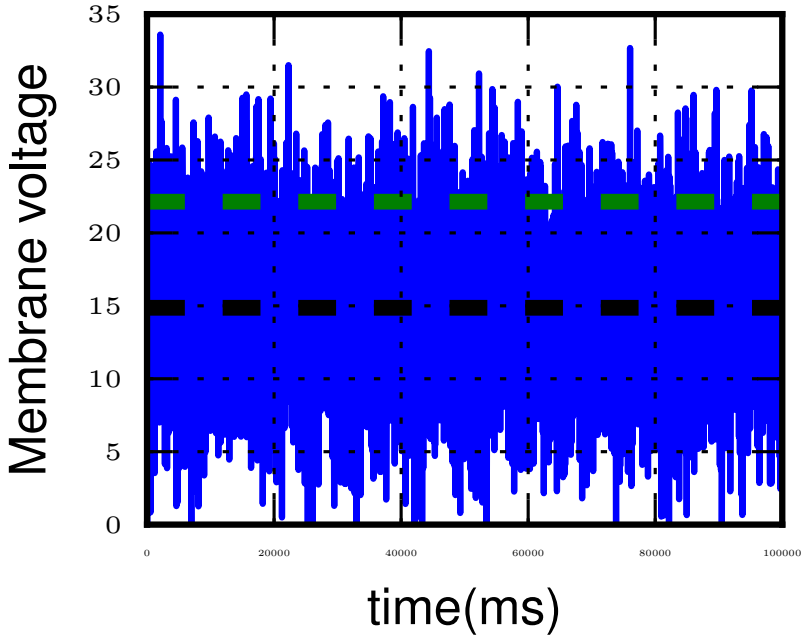
$$\mu = \frac{\tau_\alpha J e r \tau_m}{C}. \tag{4.4.18}$$

where r is the firing rate. The variance σ^2

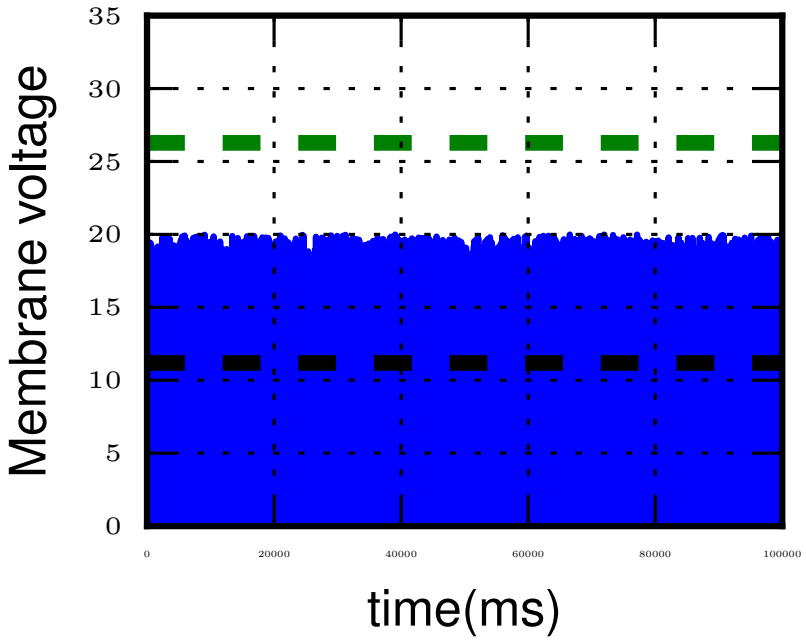
$$\begin{aligned} \sigma^2 &= \left\langle \left(y_3(t)\right)^2 \right\rangle - \left\langle y_3(t) \right\rangle^2 \\ &= \lim_{T\to\infty} \frac{1}{T} \int_0^T \left(y_3(t) - \left\langle y_3(t) \right\rangle\right)^2 dt \\ &\overset{\text{Campbell}}{=} r \int_0^\infty h^2(t)\, dt \\ &= \frac{r J^2 \tau_\alpha^2 e^2}{C^2} \int_0^\infty e^{-\frac{2t}{\tau_m}} dt \end{aligned} \tag{4.4.19}$$

$$\sigma^2 = \frac{r \tau_m}{2} \frac{J \tau_\alpha^2 e^2}{C^2}. \tag{4.4.20}$$

The overall firing rate is the number of spikes divided by the simulation duration, which can be calculated directly from the simulated data.



A



B

Figure 4.4.3: Neuron receiving input from two Poisson generators supplying excitatory and inhibitory inputs. The parameters taken here are $\tau_m = 20$ ms, $\tau_\alpha = 2$ ms, $J = 0.1$ mV, $g = 5$, $\mu = 15$ mV,$\sigma^2 = 25$ mV^2, $V_\theta = 20$ mV and $V_{\text{rest}} = 0$ mV. The overall mean and variance are adjusted to 15 mV and 25 mV^2 respectively. Simulation is performed for grid resolution $h = 5.0$ms and simulation time $t = 100$ s. Panel **(A)** shows the dynamics of membrane voltage with respect to time for the `lossless` model that has no threshold voltage. The NEST computed mean and variance are 14.86 mV and 22.14 mV respectively (here indicated by the black and green dotted lines). Panel **(B)** shows similar simulation with threshold voltage $V_\Theta = 20$ mV. The NEST computed mean and variance in this case are 11.18 mV and 26.27 mV (here indicated by the black and green dotted lines).

The NEST computed mean agrees almost exactly with the pre-set mean value in case **(A)** where there is no threshold set. However, in case **(B)** we observe a large deviation in the value of the NEST computed mean from the pre-set mean. This is attributed to the fact that whenever the voltage hits threshold, it is reset back to the resting voltage (here 0 mV) and this reset due to spiking also influences the overall dynamics of the network as seen closely in 4.4.4. This is not the case when there is no threshold set, which validates exactly the results observed analytically using the Campbell's theorem as observed in 4.4.18.

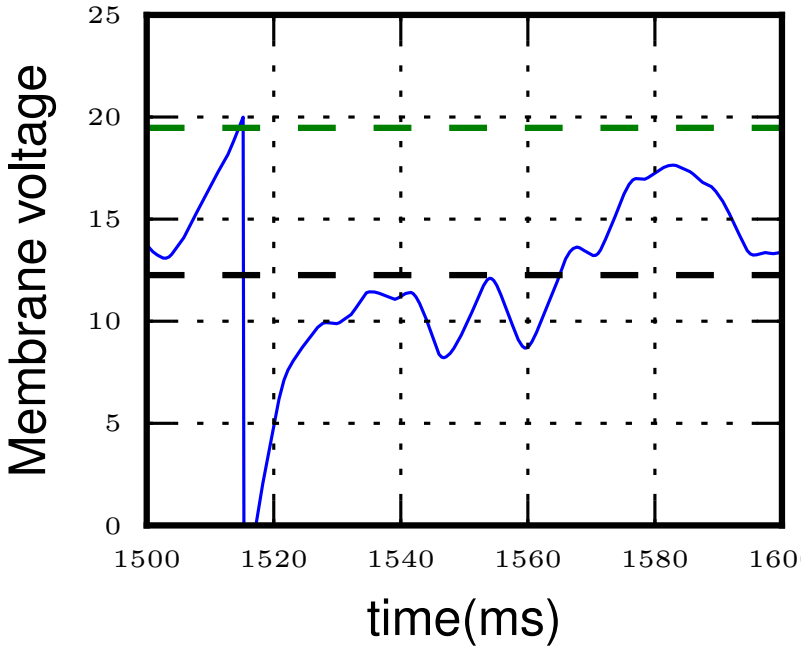


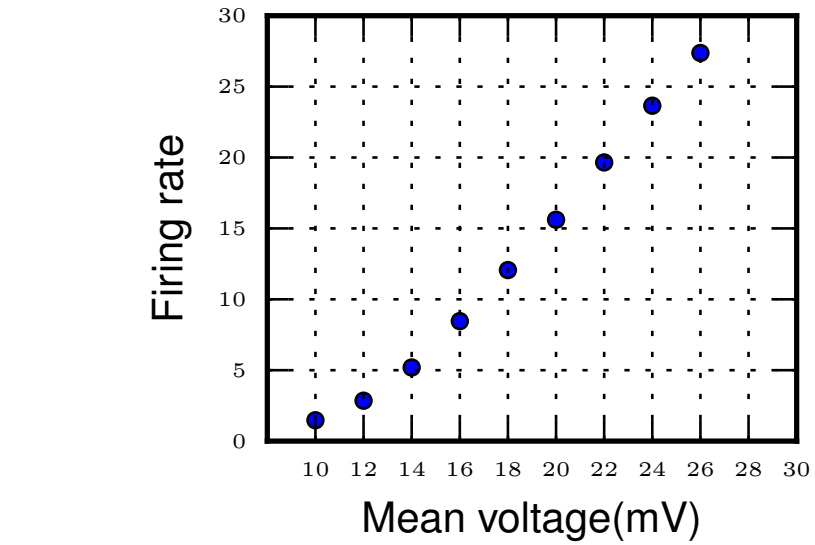
Figure 4.4.4: Dynamics of membrane voltage with respect to time for a neuron connected to two Poisson generators sending excitatory and inhibitory inputs, respectively. Zoom in of a period in time when the neuron reaches threshold at $V_\Theta = 20\,\text{mV}$. In the time interval considered here, we observe that at the time instance when this threshold is reached the neuron is reset to its resting voltage immediately, which is followed by a short period of refractoriness after which the neuron starts firing again.

This aspect of neuron dynamics is motivated here because it has an influence on the firing rate of the neuron which will be discussed in Section 4.5.

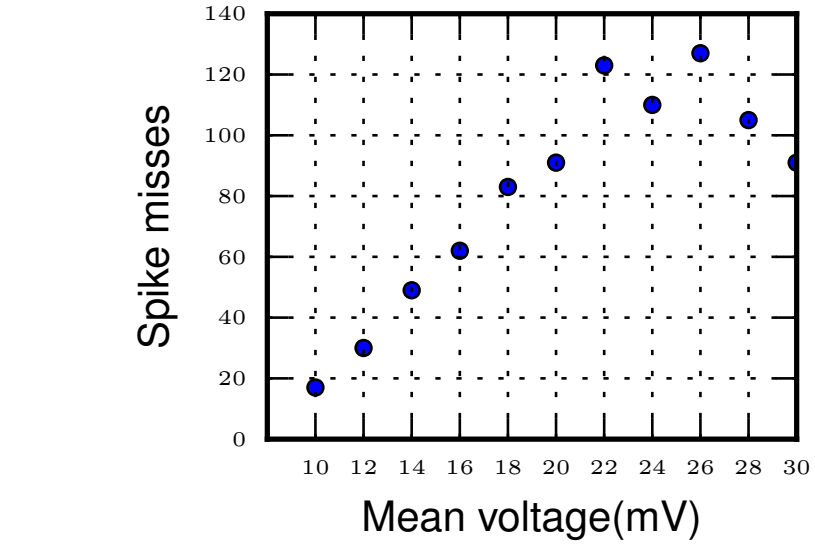
4.5 Efficiency of spike test

We now take the setting Figure 4.4.1 on page 44 of the neuron embedded in a quasi-network setup described in Section 4.4 as the basis for getting a better understanding of the advan-

tages and limitations of including the additional `spike test` function `Algorithm 4` on page 38.



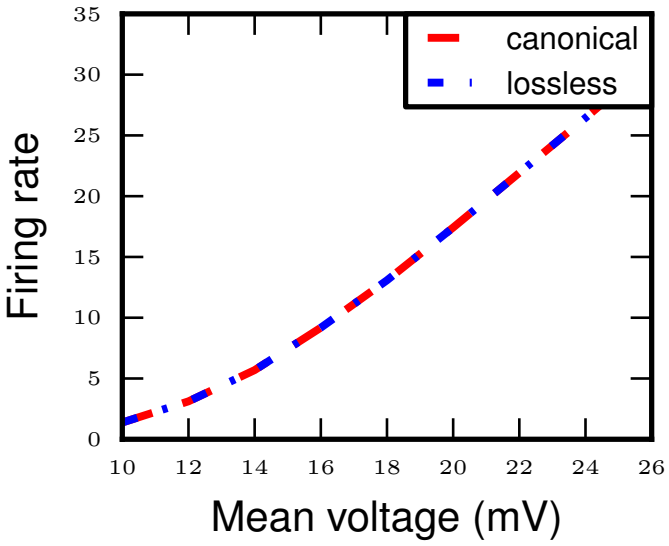
A



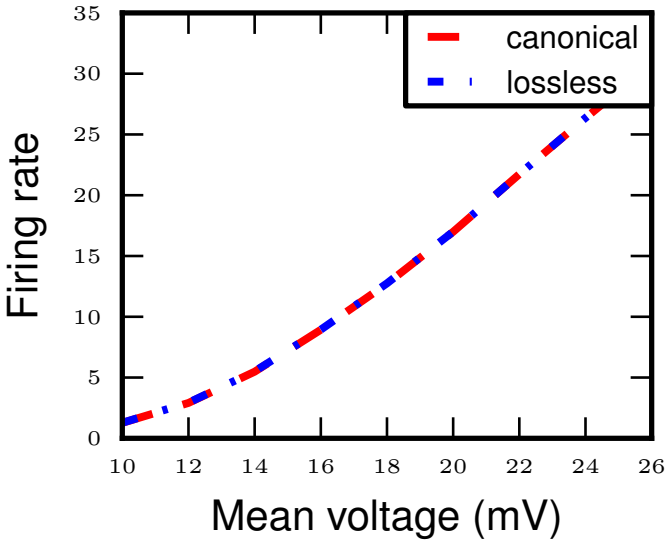
B

Figure 4.5.1: Effect of mean voltage on firing rate and spike misses. For $\mu = [10, 12 \ldots, 24]$ mV, $\sigma^2 = 25\,\text{mV}^2$, $\tau_m = 20\,\text{ms}$, $\tau_\alpha = 2\,\text{ms}$, $J = 0.1\,\text{mV}$, $g = 5$, $t = 100\,\text{s}$ and for simulation grid size $h = 5\,\text{ms}$ panel **(A)** This is verified by the plot in panel **(B)** that shows spike misses as a function of mean voltage. It should be noted here that these results are congruent both when the initial conditions were set explicitly to the case of silent neuron and also when this is not pre-set and is not part of the parameter dictionary.

Figure 4.5.1 on page 50 panel **(A)** illustrates that for increasing mean voltage the firing rate also increases which leads to the natural conclusion that spike misses also show a general rising trend with increase in mean voltage. This supports the rationale that for increasing mean voltage the chance that the voltage hits the threshold (here $V_\Theta = 20\,\text{mV}$) also increases. Intuitively one would suppose that since the firing rate increases with increase in mean voltage, the number of missed spikes will also increase monotonically. This is supported by panel **(B)** in Figure 4.5.1 on page 50. Up to 20 mV (the threshold voltage) we see that the number of spike misses adopt a linear trend. At this point there are 91 spikes that are missed within a simulation of duration $t = 100\text{sec}$. For mean voltage of 22 mV we see a sudden increase of the number of misses to 123 and the trend goes up and drops down again to 127 at 26 mV. How does one explain the sudden increase in spike misses at 22 mV? This is because the threshold V_Θ is set to a standard value of 20 mV. To check if this hypothesis is true, the threshold was set to a lower value (10 mV) it was verified (data not shown) that the number of spike misses exhibit a similar increase immediately after the membrane potential crosses threshold. The conclusion that we would draw here is that as a consequence of increasing mean voltage, firing rate and number of spike misses increases for a fixed simulation time and grid size. This motivates the investigation of effect of grid size on firing rate. For fine grid resolutions, the `canonical` and `lossless` model should show similar trends in firing rate. Ideally, there should be no spikes that are missed at this level of resolution.



A

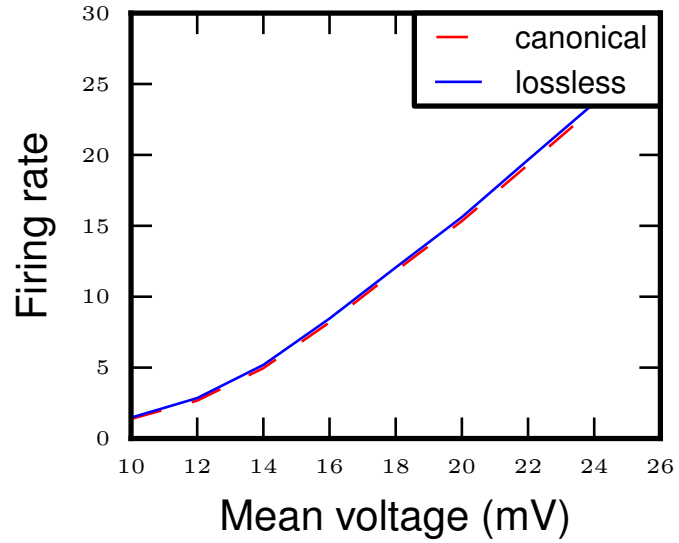


B

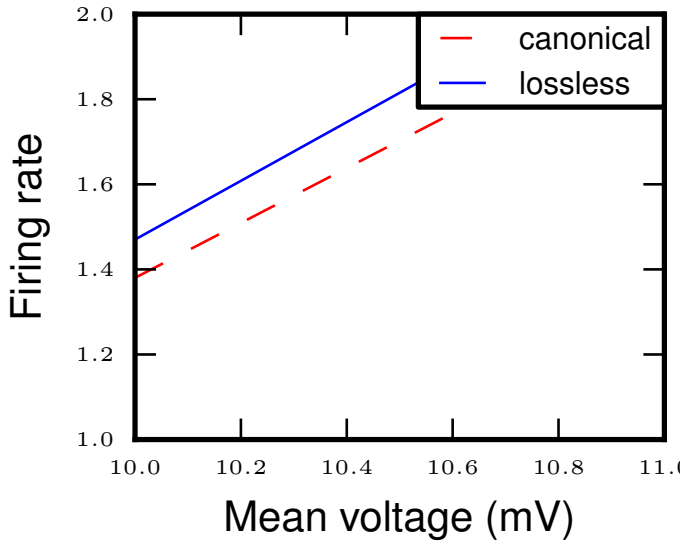
Figure 4.5.2: Comparison of firing rate as a function of mean voltage between the `canonical` (shown here by the red dashed lines) and `lossless` (shown here by the blue dashed lines) α-model for a grid resolution $h = 0.1$ ms in panel **(A)** and $h = 1.0$ ms in panel **(B)**. The other simulation parameters are $\mu = [10, 12 \ldots, 24]$ mV, $\sigma^2 = 25\,\text{mV}^2$, $\tau_m = 20$ ms, $\tau_\alpha = 2$ ms, $J = 0.1$ mV, $g = 5$ and simulation time $t = 10$ s.

From Figure 4.5.2 on page 52 one can observe that the trend in firing rate adopted by the `canonical` and `lossless` model. They overlay each other exactly thereby confirming the idea sugessted by Figure 4.3.4 on page 43 that at low resolutions there are nearly no

(∼ 7) spike misses. At this grid resolution, using the `canonical` model is rational. Now we would like to observe the trend in firing rates of the `canonical` and `lossless` model for coarser grid resolutions.



A

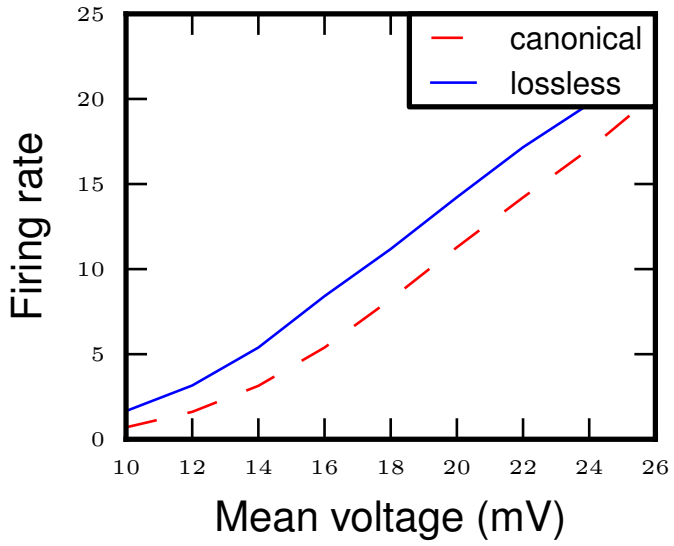


B

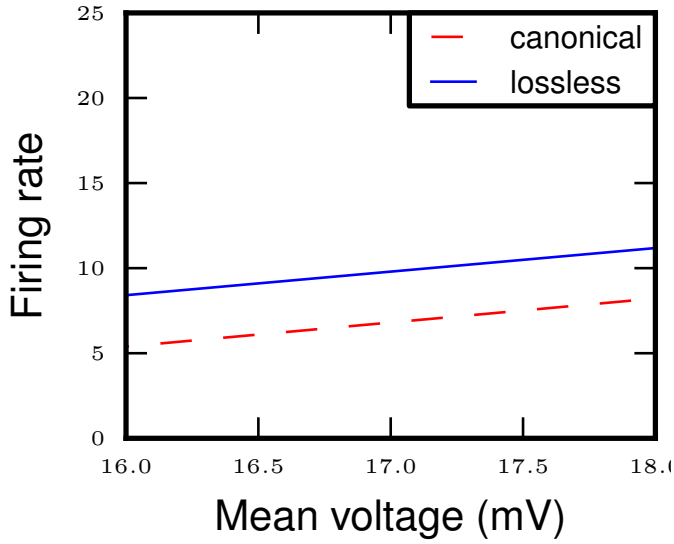
Figure 4.5.3: Comparison of firing rate as a function of mean voltage between the `canonical` (shown by the red dashed lines) and `lossless` (shown by the blue line) α-model for a grid resolution $h = 5.0$ ms. The other simulation parameters are $\mu = [10, 12 \ldots, 24]$ mV, $\sigma^2 = 25$ mV^2, $\tau_m = 20$ ms, $\tau_\alpha = 2$ ms, $J = 0.1$ mV, $g = 5$ and simulation time $t = 100$ s.

Figure 4.5.3 on page 53 illustrates that the firing rate of the `lossless` model is persis-

tently above that of the `canonical` model, suggesting that this is because the spikes missed by the `canonical` model for higher resolutions are captured by the `lossless` model. Figure 4.5.3 on page 53 Panel **(A)** shows that there is a small but distinguishable difference between the firing rate trend of the `canonical` and `lossless` model which can be caught better in panel **(B)**, where we observe that this is ~ 0.2 Hz. This difference in firing rate corresponds to ~ 50 spike misses as can be verified in Figure 4.5.5 on page 56. Here we draw attention to the observation made from Figure 4.4.4 on page 49, where the aspect that the membrane voltage is reset immediately to the resting voltage whenever it hits the threshold was discussed. Continuous voltage resets after spiking also influences firing rate.



A



B

Figure 4.5.4: Comparison of firing rate as a function of mean voltage between the `canonical` (shown by the red dashed lines) and `lossless` (shown by the blue line) α-model for a grid resolution $h = 5.0$ ms. The other simulation parameters are $\mu = [10, 12 \ldots, 24]$ mV, $\sigma^2 = 25\,\text{mV}^2$, $\tau_m = 20$ ms, $\tau_\alpha = 2$ ms, $J = 0.1$ mV, $g = 5$ and $t = 100$ s. Panel **(A)** shows that there is a defined difference in firing rate between the `canonical` and `lossless` model which from panel **(B)** is observed to be equivalent to ~ 4 Hz.

Figure 4.5.4 on page 55 shows that there is a consistently higher firing rate in the `lossless` model compared to the `canonical` model. This suggests that the spike misses detected by

the `spike_test Algorithm 4` in the `lossless` model are counterbalanced by emitting spikes Algorithm 2 thereby exhibiting higher firing rates. This corresponds to ~ 400 spike misses as verified in Figure 4.5.5 on page 56.

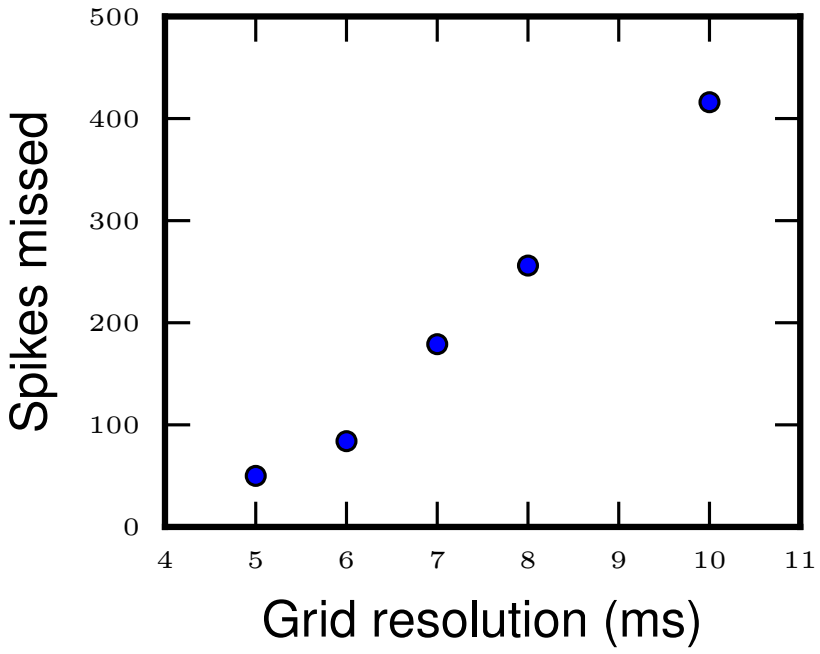


Figure 4.5.5: Spike misses for a range of grid resolution for a simulation time of $t = 100\,\mathrm{s}$ The number of spikes missed increases with larger grid spacing. The other simulation parameters are $\mu = 15\,\mathrm{mV}$, $\sigma^2 = 25\,\mathrm{mV}^2$, $\tau_m = 20\,\mathrm{ms}$, $\tau_\alpha = 2\,\mathrm{ms}$, $J = 0.1\,\mathrm{mV}$ and $g = 5$.

Figure 4.5.5 on page 56 shows that the number of spike misses increases with increase in grid spacing. The number of spike misses here corresponds approximately to the higher firing rate that we observe in the `lossless` model as seen in Figure 4.5.3 on page 53 and Figure 4.5.4 on page 55. We also make the natural observation that the number of spike misses increases in proportion to the simulation time.

So far it has been motivated that the `lossless` model is efficient to capture the missed spikes since the `spike_test` computes the precise spike timing [Morrison et al., 2005a] using the Lambert W function [Corless et al., 1996]. However this also indicates that such a computation comes with a cost. Therefore it is worthy to investigate the execution time of `lossless` model and compare it with the `canonical` α-model.

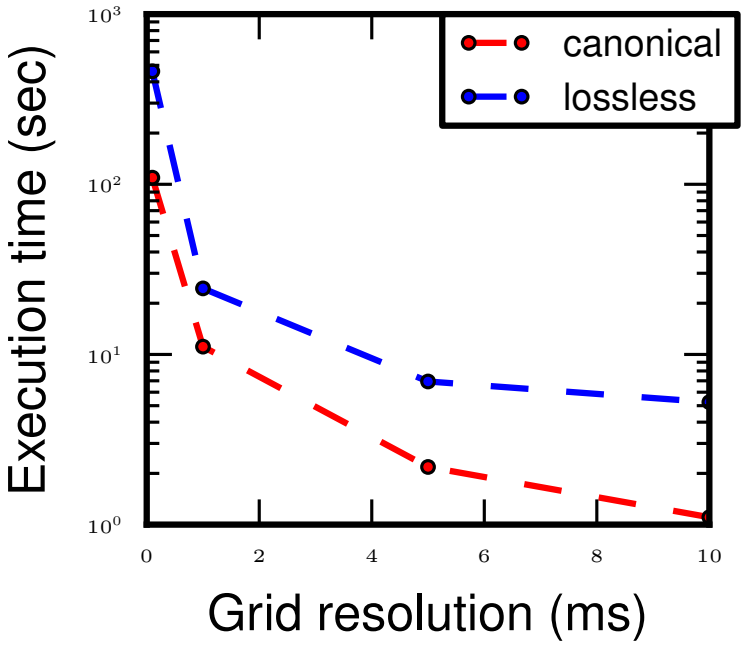


Figure 4.5.6: Execution time as a function of grid resolution (h = 0.1 ms,1.0 ms, 5.0 ms and 10.0 ms) for a simulation time $t = 10$ s. The blue line shows the time taken by the `lossless` model and the red line by the `canonical` model. The other simulation parameters are $\mu = 15$ mV, $\sigma^2 = 25$ mV^2, $\tau_m = 20$ ms, $\tau_\alpha = 2$ ms, $J = 0.1$ mV and $g = 5$.

From Figure 4.5.6 on page 57 we make two major observations. The execution time decreases with increase with the grid spacing since the number of computation steps are lesser. However, it is also seen that the `lossless` model takes continuously more time as compared to the canonical model, even for smaller grid sizes. This is attributed to the fact that every time there is an incoming event the `lossless` model has to make a yes or no decision by executing the `spike_test Algorithm 4` (and if yes, go find the threshold crossing by interpolation), which happens at the expense of computation time.

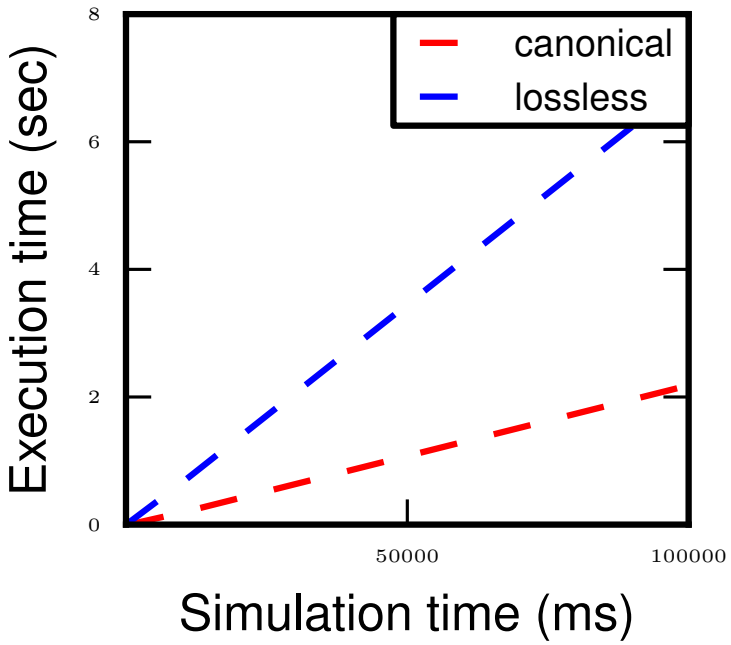


Figure 4.5.7: Execution time as a function of simulation time for the standard grid size $h = 5$ ms. The blue line shows the trend for the `lossless` model and red line for the `canonical` model. The other simulation parameters are $\mu = 15$ mV, $\sigma^2 = 25$ mV^2, $\tau_m = 20$ ms, $\tau_\alpha = 2$ ms, $J = 0.1$ mV and $g = 5$.

Figure 4.5.7 on page 58 validates the observations made from Figure 4.5.6 on page 57. It shows that the execution time increases linearly with simulation duration. The time taken for execution by the `lossless` model is consistently more than that of the `canonical` model i.e. the `lossless` model is efficient but computationally expensive. This suggests that it is rational to use the `canonical` model for grid resolutions < 5 ms and choose between the `canonical` or `lossless` model for grid sizes above this depending on the efficiency that is desired by the user.

5 Discussion

The `lossless` model is the accurate implementation of the mathematical model describing the LIF neuron with α-PSCs. The model is efficient and can catch otherwise missed spikes in coarse grid resolutions Figure 4.5.5 on page 56. Since the `lossless` model never looses spikes its accuracy is always above that of the `canonical` model. We also observe from the comparison of execution time between the two models Figure 4.5.7 on page 58 that this accuracy comes with computational cost.

However, it is rational to compare the execution time of two models that have equivalent accuracy which was not the case in our preliminary analysis here. Therefore a comparison of the canonical α-model with a pre-set tolerance level for spike misses with that of the lossless α-model would give us a better insight into the efficiency of the new model.

A detailed analysis of the state variables governing the system of equations of the model could help us exclude more initial conditions that correspond to the situation where there can be no spike misses, thereby optimizing the `spike test` function and improving the efficiency of the `lossless` model.

Bibliography

Rajagopal Ananthanarayanan and Dharmendra S. Modha. Anatomy of a cortical simulator. In *Supercomputing 2007: Proceedings of the ACM/IEEE SC2007 Conference on High Performance Networking and Computing*, New York, NY, 2007. Association for Computing Machinery.

C. Bernard, Y. C. Ge, E. Stockley, J. B. Willis, and H. V. Wheal. Synaptic integration of NMDA and non-NMDA receptors in large neuronal network models solved by means of differential equations. *Biological Cybernetics*, 70:267–273, 1994.

Norman Campbell. The study of discontinuous phenomena. *ProcCambridgePhilSoc*, 15: 117–136, 1909.

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

Markus Diesmann and Marc-Oliver Gewaltig. NEST: An environment for neural systems simulations. In Theo Plesser and Volker Macho, editors, *Forschung und wisschenschaftliches Rechnen, Beiträge zum Heinz-Billing-Preis 2001*, volume 58 of *GWDG-Bericht*, pages 43–70. Ges. für Wiss. Datenverarbeitung, Göttingen, 2002.

Markus Diesmann, Marc-Oliver Gewaltig, Stefan Rotter, and Ad Aertsen. State space analysis of synchronous spiking in cortical neural networks. *Neurocomputing*, 38–40:565–571, 2001.

Mikael Djurfeldt. *Large-scale Simulation of Neuronal Systems*. PhD thesis, KTH School of Computer Science and Communication, Stockholm, Sweden, 2009.

J. M. Eppler, M. Helias, E. Muller, M. Diesmann, and M. Gewaltig. PyNEST: a convenient interface to the NEST simulator. 2:12, 2009.

Alois Ferscha. Parallel and distributed simulation of discrete event systems. In Albert Y. Zomaya, editor, *Parallel and Distributed Computing Handbook*, chapter 35, pages 1003–1041. McGraw-Hill, 1996.

Richard M. Fujimoto. *Parallel and distributed simulation systems*. Wiley, New York, 2000. ISBN 0-471-18383-0.

Marc-Oliver Gewaltig and Markus Diesmann. NEST (NEural Simulation Tool). *Scholarpedia*, 2(4):1430, 2007.

Sven Goedeke and Markus Diesmann. The mechanism of synchronization in feed-forward neuronal networks. 10:015007, 2008.

D. F. M. Goodman and R. Brette. The Brian simulator. 3(2):192–197, 2009. doi: 10.3389/neuro.01.026.2009.

D. Hansel, G. Mato, C. Meunier, and L. Neltner. On numerical simulations of integrate-and-fire neural networks. *Neural Computation*, 10(2):467–483, Feb., 15 1998.

Moritz Helias, Tom Tetzlaff, and Markus Diesmann. Echoes in correlated neural systems. 15:023002, 2013.

Michael Hines. NEURON — a program for simulation of nerve equations. In F. Eeckman, editor, *Neural Systems: Analysis and Modeling*, pages 127–136. Kluwer Academic Publishers, Norwell, Ma, 1993.

Susanne Kunkel, Moritz Helias, Markus Diesmann, and Abigail Morrison. Fail-safe detection of threshold crossings of linear integrate-and-fire neuron models in time-driven simulations. *BMC Neuroscience*, 12(Suppl 1):P229, 2011.

A. Morrison, A. Aertsen, and M. Diesmann. Spike-timing dependent plasticity in balanced random networks. *Neural Computation*, 19:1437–1467, 2007a.

A. Morrison and M. Diesmann. Maintaining causality in discrete time neuronal network simulations. In P. beim Graben, C. Zhou, M. Thiel, and J. Kurths, editors, *Lectures in Supercomputational Neuroscience: Dynamics in Complex Brain Networks*, Understanding Complex Systems, pages 267–278. Springer, 2008.

Abigail Morrison, Johan Hake, Sirko Straube, Hans Ekkehard Plesser, and Markus Diesmann. Precise spike timing with exact subthreshold integration in discrete time network simulations. In *Proceedings of the 30th Göttingen Neurobiology Conference*, page 205B, 2005a.

Abigail Morrison, Carsten Mehring, Theo Geisel, Ad Aertsen, and Markus Diesmann. Advancing the boundaries of high connectivity network simulation with distributed computing. *Neural Computation*, 17(8):1776–1801, 2005b.

Abigail Morrison, Sirko Straube, Hans Ekkehard Plesser, and Markus Diesmann. Exact subthreshold integration with continuous spike times in discrete time neural network simulations. *Neural Computation*, 19(1):47–79, 2007b.

H. E. Plesser and M. Diesmann. Simplicity and efficiency of integrate-and-fire neuron models. *Neural Computation*, 21:353–359, 2009.

Python Software Foundation. The Python programming language, 2008. http://www.python.org.

A. Rauch, G. La Camera, H. Lüscher, W. Senn, and S. Fusi. Neocortical pyramidal cells respond as integrate-and-fire neurons to in vivo like input currents. *Journal of Neurophysiology*, 90:1598–1612, 2003.

Stefan Rotter and Markus Diesmann. Exact digital simulation of time-invariant linear systems with applications to neuronal modeling. *Biological Cybernetics*, 81(5/6):381–402, 1999.

M. J. Shelley and L. Tao. Efficient and accurate time-stepping schemes for integrate-and-fire neuronal networks. *Journal of Computational Neuroscience*, 11(2):111–119, 2001.

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

Matthew A. Wilson and James M. Bower. The simulation of large-scale neural networks. In Christof Koch and Idan Segev, editors, *Methods in Neuronal Modeling: From Synapses to Networks*, chapter 9, pages 291–333. A Bradford Book, The MIT Press, Cambridge, Massachusetts, 1989.

Bernard P. Zeigler, Herbert Praehofer, and Tag Gon Kim. *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press, Amsterdam, 2 edition, 2000.