



# Probabilistic solvers enable a straight-forward exploration of numerical uncertainty in neuroscience models

Jonathan Oesterle<sup>1</sup> · Nicholas Krämer<sup>2</sup> · Philipp Hennig<sup>2,3,4</sup> · Philipp Berens<sup>1,4</sup>

Received: 15 December 2021 / Revised: 14 May 2022 / Accepted: 2 July 2022 / Published online: 6 August 2022  
© The Author(s) 2022

## Abstract

Understanding neural computation on the mechanistic level requires models of neurons and neuronal networks. To analyze such models one typically has to solve coupled ordinary differential equations (ODEs), which describe the dynamics of the underlying neural system. These ODEs are solved numerically with deterministic ODE solvers that yield single solutions with either no, or only a global scalar error indicator on precision. It can therefore be challenging to estimate the effect of numerical uncertainty on quantities of interest, such as spike-times and the number of spikes. To overcome this problem, we propose to use recently developed sampling-based probabilistic solvers, which are able to quantify such numerical uncertainties. They neither require detailed insights into the kinetics of the models, nor are they difficult to implement. We show that numerical uncertainty can affect the outcome of typical neuroscience simulations, e.g. jittering spikes by milliseconds or even adding or removing individual spikes from simulations altogether, and demonstrate that probabilistic solvers reveal these numerical uncertainties with only moderate computational overhead.

**Keywords** Probabilistic numerics · Computational neuroscience · Uncertainty quantification · Hodgkin–Huxley

## 1 Introduction

Computational neuroscience is built around computational models of neurons that allow the simulation and analysis of signal processing in the central nervous system. These models can describe neural computations on different levels of abstraction. On the *statistical level*, e.g. generalized linear models have been used to provide a probabilistic model mapping environmental variables to neural activity (Pillow et al., 2008). For such statistical models, quantifying the uncertainty of the parameters can be achieved using Bayesian approaches (Gerwinn et al., 2008). On the *mechanistic level*,

the models typically take the form of systems of coupled ordinary differential equations (ODEs), which describe the dynamics of the membrane potential and give rise to the spike-times (Gerstner & Kistler, 2002; Izhikevich, 2007). Recently, likelihood-free inference approaches have made it possible to perform uncertainty-aware inference even for such complicated mechanistic models (Gonçalves et al., 2020; Oesterle et al., 2020; Papamakarios et al., 2018).

However, mechanistic models of neurons are subject to an additional source of uncertainty: the numerical error caused by the solution of the model's ODEs with a concrete algorithm (Hennig et al., 2015). This arises because all numerical solvers are necessarily run with finite time and limited resources, so their estimate diverges from the true solution of the ODE, even if the problem is well-posed. When simulating neurons, one would like to compute a numerical solution close to the true solution of the ODE, to ensure that conclusions drawn from the simulations are based on the mechanisms described by the model rather than the specific choice, setting and implementation of the ODE solver.

Many of the well-established numerical solvers do report a global error estimate and a corresponding tolerance that can be set by the user (Hairer et al., 1993, Chapter II.4). This global scalar error, though, does not capture how the numerical error arising from finite step-sizes used in practice

Action Editor: David Golomb

✉ Philipp Berens  
philipp.berens@uni-tuebingen.de

<sup>1</sup> Institute of Ophthalmic Research, University of Tübingen, Tübingen, Germany

<sup>2</sup> Department of Computer Science, University of Tübingen, Tübingen, Germany

<sup>3</sup> Max Planck Institute for Intelligent Systems, Tübingen, Germany

<sup>4</sup> Tübingen AI Center, University of Tübingen, Tübingen, Germany

affects crucial quantities of interest in the simulation, such as spike-times or the number of spikes. In practice, it can therefore be challenging to select a tolerance that strikes a good balance between run time and accuracy.

For some of the most common mechanistic models in neuroscience like the Hodgkin–Huxley or Izhikevich neuron model, errors in numerical integration have been studied in detail for a range of solvers and different integration step-sizes (Stewart & Bair, 2009; Borgers & Nectow, 2013; Chen et al., 2020). These studies have shown that standard solvers are often not the best choice in terms of accuracy or the accuracy vs. run time tradeoff. Therefore, the authors of these studies proposed to use specific solvers for the analyzed models, e.g. the Parker–Sochacki method for the Hodgkin–Huxley and Izhikevich neuron (Stewart & Bair, 2009), an exponential midpoint method (Borgers & Nectow, 2013) or second-order Strang splitting (Chen et al., 2020) for Hodgkin–Huxley-like models. While improving computations for the specific problems, applying these to other scenarios requires a detailed understanding of the kinetics of the neuron model of interest; and while choosing a “good” solver for a given model is important, it is typically not necessary to choose the “best” ODE solver. In many cases, it can be sufficient to ensure that the computed solution is within a certain accuracy.

As a more general approach to quantify the numerical uncertainty in mechanistic models in neuroscience, we therefore propose to use probabilistic ODE solvers (Hennig et al., 2015; Oates & Sullivan, 2019; Cockayne et al., 2019). In contrast to classical ODE solvers, this class of solvers does not only yield a single solution, but instead a distribution over solutions that quantifies the numerical uncertainty.

Several frameworks for probabilistic ODE solvers have been proposed, which differ mostly in the tradeoff between computational cost and flexibility of the posterior, from fast Gaussian filters (Schober et al., 2019; Tronarp et al., 2019; Krämer et al., 2021) to sampling-based approaches (Conrad et al., 2017; Chkrebtti et al., 2016; Teymur et al., 2016, 2018; Abdulle & Garegnani, 2020). These solvers have been mostly tested for well-behaved systems with well-behaved solutions, but the ODEs used to simulate neural activity model the non-linear membrane dynamics that underlie the all-or-none nature of an action potential. Here, we use two related approaches of probabilistic ODE integration, the state perturbation proposed by Conrad et al. (2017) and the step-size perturbation of Abdulle and Garegnani (2020). Both build on existing explicit, iterative ODE solvers and stochastically perturb the numerical integration of individual steps taken by the underlying solvers. These perturbations make the solution of every step probabilistic and therefore also the solution as a whole. The magnitude of the perturbation has to be calibrated, such that the solver’s output distribution reflects the numerical uncertainty in the solution.

Here, we explore the potential of probabilistic ODE solvers for neuron models. We show how the state and step-size perturbation methods can be used to quantify and reveal numerical uncertainty caused by the numerical ODE integration and demonstrate that the solver outputs are easy to interpret. For this, we simulate typical neuron models, namely the Izhikevich neuron model (Izhikevich, 2004), as a representative of leaky-integrate-and-fire neuron models, single-compartment Hodgkin–Huxley models (Hodgkin & Huxley, 1952) and a model with three synaptically coupled Hodgkin–Huxley-like neurons (Prinz et al., 2004) as an example of a neuronal network model. Lastly, we discuss practical considerations and limitations of these probabilistic solvers such as the calibration of the perturbation and the computational overhead.

Taken together, our results suggest that probabilistic ODE solvers should be considered as a useful tool for the simulation of neuronal systems, to increase the quality and reliability of such simulations over those achieved with classic solvers.

## 2 Methods and models

### 2.1 Probabilistic solvers

Simulating neuron models typically amounts to solving an initial value problem (IVP) based on a set of coupled ODEs. In abstract form, an initial value problem is given by

$$\dot{\mathbf{x}}(t) = f(t, \mathbf{x}(t)), \quad \mathbf{x}(t_0) = \mathbf{x}_0, \quad (1)$$

where  $f$ ,  $\mathbf{x}_0$  and  $t_0$  are known and  $\mathbf{x}(t)$  for  $t > t_0$  is the quantity of interest. The solution to the initial value problem at time  $t + \Delta t$  provided the solution at time  $t$ , is given by integrating Eq. (1) from  $t$  to  $t + \Delta t$ :

$$\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + \int_t^{t+\Delta t} f(s, \mathbf{x}(s)) ds. \quad (2)$$

Except for special cases, this integral has no analytic form and must be solved numerically. For example, the forward Euler method approximates the integral as  $\int_t^{t+\Delta t} f(s, \mathbf{x}(s)) ds \approx \Delta t \cdot f(t, \mathbf{x}(t))$ . To simulate a neuron, Eq. (2) is solved iteratively, which results in a sequence of solutions  $X = [\mathbf{x}(t_0), \mathbf{x}(t_1), \mathbf{x}(t_2), \dots, \mathbf{x}(t_M)]$  for a set of time points with  $t_{i+1} > t_i$  and a maximum time point  $t_M$ . Standard solvers yield a deterministic solution in every step, and therefore for the solution  $X$  as a whole. In contrast, the probabilistic solvers used in this study stochastically perturb the numerical integration used to approximate Eq. (2), which makes the solution of every step—and therefore of the whole solution—probabilistic. For a given IVP and solver, one can therefore generate a sample distribution of solutions  $X$  by repeating the iterative numerical integration from  $t_0$  to  $t_M$  multiple times.

To create these probabilistic solvers, we implemented the state perturbation algorithm of Conrad et al. (2017) and the step-size perturbation algorithm of Abdulle and Garegnani (2020).

In the state perturbation algorithm (Conrad et al., 2017), in each step of the numerical integration, a small independently drawn noise term  $\xi_t$  is added to the solution  $\mathbf{x}_{\text{det.}}(t + \Delta t)$  of a corresponding deterministic integration scheme:

$$\mathbf{x}_{\text{prb}}(t + \Delta t) = \mathbf{x}_{\text{det.}}(t + \Delta t) + \xi_t, \tag{3}$$

$$\xi_t \sim \mathcal{N}(\mathbf{0}, \text{diag}(\mathbf{v}_t)^2),$$

where  $\mathbf{v}_t$  controls the magnitude of the perturbation. The perturbation is only efficient when  $\mathbf{v}_t$  is of the right order: if chosen too small, the uncertainty will be underestimated; if chosen too large, it will render the solver output useless. Conrad et al. (2017) suggested calibrating  $\mathbf{v}_t$  to replicate the amount of error introduced by the numerical scheme. We chose  $\mathbf{v}_t = \sigma \epsilon_t$  using the error estimator  $\epsilon_t$  readily available in methods that were developed for step-size adaptation (see Sect. 5.1), and a scalar perturbation parameter  $\sigma$  that can be adjusted to calibrate the perturbation. If not stated otherwise, we used  $\sigma = 1$ . An example of this perturbation method is shown in Fig. 1A for a single integration step and in Fig. 1C for an Izhikevich neuron model.

A related approach to stochastically perturbing the numerical integration was proposed by Abdulle and Garegnani (2020), where noise is added to the integration step-size (i.e. to the “input” of the solver, rather than the “output”, cf. Fig. 1B). The numerical integration is performed using the perturbed step-size  $\zeta_t$ , but the computed solution is treated as the solution for the original step-size  $\Delta t$ :

$$\mathbf{x}_{\text{prb}}(t + \Delta t) = \mathbf{x}_{\text{det.}}(t + \zeta_t), \quad \zeta_t \sim \mathcal{P}, \tag{4}$$

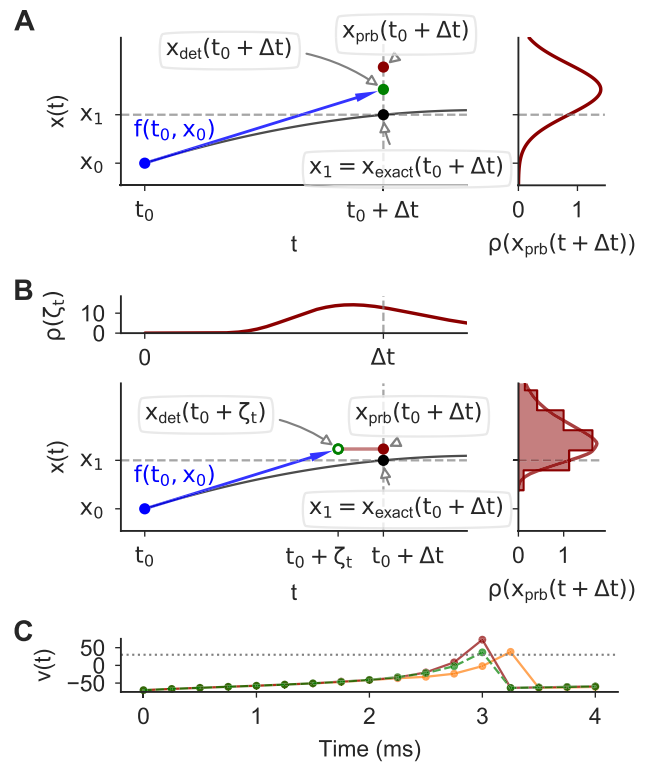
where  $\zeta_t$  is the perturbed step-size drawn from a distribution  $\mathcal{P}$  and  $\mathbf{x}_{\text{det.}}(\bullet)$  is a deterministic integration scheme that approximates Eq. (2). For example, for the forward Euler method Eq. (4) would be computed as  $\mathbf{x}_{\text{prb}}(t + \Delta t) = \mathbf{x}_{\text{det.}}(t) + \zeta_t \cdot f(t, \mathbf{x}_{\text{det.}}(t))$ . Abdulle and Garegnani (2020) defined three properties the i.i.d. random variables  $\zeta_t$  should fulfill:

- $\mathcal{P}(\zeta_t > 0) = 1$ ,
- there exists  $\Delta t$  such that  $\mathbb{E}[\zeta_t] = \Delta t$ , and
- there exist  $p \geq 0.5$  and  $C > 0$  independent of  $t$  such that  $\mathbb{E}[(\zeta_t - \Delta t)^2] = C \cdot \Delta t^{2p+1}$ .

Based on these restrictions, they proposed, as an example, to use a log-normal distribution:

$$\zeta_t \sim \mathcal{LN}_t(m, s^2), \tag{5}$$

where  $m$  and  $s$  are the mean and standard deviation of the underlying normal distribution, which are given as:



**Fig. 1 Illustration of probabilistic ODE solvers.** **A** *Left*: A single step with a state perturbed forward Euler (FE) method (Conrad et al., 2017) for the ODE  $f(t, x(t)) = 3 \cdot x(t) \cdot \sin(t + 3)$  and the exact solution  $x(t) = \exp(-3 \cdot \cos(t + 3))$  (black curve). We set  $t_0 = 0$  and the step-size to  $\Delta t = 0.1$ . The exact solution at  $t = t_0 + \Delta t$  is highlighted (black dot). A first order solution is computed using FE:  $x_{\text{det.}}(t + \Delta t) = x_0 + \Delta t \cdot f(t_0, x_0)$  ( $x_{\text{det.}}$ : green dot,  $f$ : blue arrow). *Right*: The probability density function  $\rho$  of  $x_{\text{prb}}(t + \Delta t)$ , where  $x_{\text{prb}}(t + \Delta t)$  is the output of the probabilistic step. In the state perturbation,  $\rho$  is a normal distribution with mean  $x_{\text{det.}}(t + \Delta t)$  and a standard deviation based on a local error estimator (see Sect. 2.1). A sample is shown for illustration (red dot). **B** As in A, but for the step-size perturbation (Abdulle & Garegnani, 2020). Instead of integrating from  $t_0$  to  $t_0 + \Delta t$ , the ODE is integrated from  $t_0$  to  $t_0 + \zeta_t$ , where  $\zeta_t$  is randomly drawn from a log-normal distribution (top panel). The solution of this perturbed integration  $x_{\text{det.}}(t + \zeta_t)$  (green circle) is then used as the solution  $x_{\text{prb}}(t + \Delta t)$  of the probabilistic step (red dot), making  $x_{\text{prb}}(t + \Delta t)$  a random variable with a distribution  $\rho(x_{\text{prb}}(t + \Delta t))$  (right panel), that has no general analytical form and is therefore shown as a histogram and a kernel density estimate. **C** Simulations of an Izhikevich neuron with a deterministic (green) and a state perturbed FE method (two samples: red and orange)

$$m = \ln(\Delta t^2 / \phi),$$

$$s = \sqrt{2 \ln(\phi / \Delta t)}, \tag{6}$$

$$\phi = \sqrt{\mathbb{E}^2[\zeta_t] + \text{Var}[\zeta_t]} = \sqrt{\Delta t^2 + C \cdot \Delta t^{2p+1}}.$$

Using  $p \leq O$ , where  $O$  is the order of the method, ensures that the mean-squared convergence order of the method is not changed. We used  $p = O$  throughout to maximize the effect

of the perturbation without changing the convergence order. We further generalized the example provided by Abdulle and Garegnani (2020) in which  $C = 1$  to a parametrized distribution by setting  $C = \sigma^2$ , i.e. setting  $\phi = \sqrt{\Delta t^2 + \sigma^2 \cdot \Delta t^{2O+1}}$ . The introduction of the perturbation parameter  $\sigma$  allows to—similarly to the perturbation parameter used in the state-perturbation—adjust and calibrate the magnitude of perturbation. If not stated otherwise, we used  $\sigma = 1$ . The perturbation of a single step is illustrated in Fig. 1B.

### 2.2 Choice of solvers

We used the perturbation methods described above to create probabilistic versions of the solvers listed in Table 1.

The usage of fixed (f) and adaptive (a) step-sizes is indicated with subscripts, and the perturbation method is indicated using the superscripts— $x$  for the state perturbation (Conrad et al., 2017) and  $t$  for the step-size perturbation (Abdulle & Garegnani, 2020)—meaning that e.g.  $FE_f^x$  is referring to a forward Euler method using fixed step-sizes and the state perturbation. For the exponential integrators, we chose to only use the step-size perturbation because it preserves the important property of these solvers that the activation and inactivation variables cannot leave the interval  $[0, 1]$ , and also because there are no established methods for local error estimation for these methods.

The second order exponential integrator EEMP was implemented based on the version by B3rgers and Nectow (2013) (Sect. 5.2), which is a modification of the midpoint method by Oh and French (2006). Computation of Runge–Kutta steps and step-size adaptation were based on the respective `scipy` implementations (Virtanen et al., 2020). To avoid computational overhead, we only computed the local error estimates when necessary, i.e. for adaptive step-sizes or the state perturbation.

### 2.3 Interpolation

The iterative solvers used in this study yield solutions for  $\mathbf{x}(t)$  on either a fixed and equidistant grid of time points  $T$  or, in the case

of adaptive step-size solvers, on a finite set of time points  $T$  automatically chosen by the solver. To interpolate these solutions for example for spike-time estimation (see Sect. 2.4), we used linear interpolation for FE, EE and EEMP between solutions of single steps. To interpolate the steps of the Runge–Kutta methods we utilized the “dense output” implemented in the respective `scipy` methods (Virtanen et al., 2020). These “dense outputs” allow evaluating the solution between two steps  $\mathbf{x}(t_i)$  and  $\mathbf{x}(t_{i+1})$  for any  $t$  with  $t_i \leq t \leq t_{i+1}$  without any additional ODE evaluation. To not discard the effect of the state perturbation during interpolation, we defined the dense output  $\hat{d}_{RK}(t, t_i, t_{i+1})$  for a state perturbed Runge–Kutta step from time  $t_i$  to  $t_{i+1}$  as:

$$\hat{d}_{RK}(t, t_i, t_{i+1}) = d_{RK}(t, t_i, t_{i+1}) + \frac{t - t_i}{t_{i+1} - t_i} \xi_{t_i}, \tag{7}$$

where  $d_{RK}(t, t_i, t_{i+1})$  is the dense output of the respective deterministic Runge–Kutta step and  $\xi_{t_i}$  is the perturbation noise that was added to this step to compute  $\mathbf{x}(t_{i+1})$  (see Eq. (3)). This is a simplified version of the continuous-time output proposed by Conrad et al. (2017).

### 2.4 Spike-time estimation

To determine spike-times based on simulated voltage traces  $v(t)$ , we interpolated the ODE solutions for all steps where  $v(t)$  started from below and ended above a certain threshold voltage  $v_{th}$ . For linearly interpolated solutions (Sect. 2.3) we computed spike-times as follows. For every step from a time  $t_i$  to  $t_{i+1}$  with  $v(t_i) < v_{th} \leq v(t_{i+1})$  we estimated the respective spike-time  $t_{spike}$  as:

$$t_{spike} = \frac{v_{th} - v(t_i)}{t_{i+1} - t_i}. \tag{8}$$

To estimate spike-times for Runge–Kutta methods with “dense-outputs”, we utilized `scipy`’s “brentq” root finding algorithm to determine the time point  $t_{spike}$  when the threshold is reached, i.e.  $|v(t_{spike}) - v_{th}| < \epsilon$ , with  $\epsilon = 1e - 12$ .

**Table 1** Summary of the ODE solvers used in this paper

Abbr.	$O$	$O_e$	Method & Error estimate
FE	1	2	Forward Euler with Heun’s method for error estimation.
EE	1		Exponential Euler.
EEMP	2		Exponential Euler Midpoint (Borgers & Nectow, 2013).
RKBS	3	2	Bogacki–Shampine, an embedded Runge–Kutta method (Bogacki & Shampine, 1989).
RKCK	4	5	Cash–Karp method, an embedded Runge–Kutta method (Cash & Karp, 1990).
RKDP	5	4	Dormand–Prince, an embedded Runge–Kutta method (Dormand & Prince, 1980).

$O$  and  $O_e$  are the orders of the solution and the error estimator, respectively. See Sect. 5.2 for details

## 2.5 Common ODE models in computational neuroscience

In this study, we use probabilistic ODE solvers to analyze the effect of numerical uncertainty in the following neuroscience models:

- The Izhikevich neuron model with a range of dynamics,
- the Hodgkin–Huxley neuron model,
- and a small network of Hodgkin–Huxley neurons.

We picked these models to cover both single neuron models and models of neuronal networks.

### 2.5.1 Single Izhikevich neurons

The Izhikevich neuron (IN) model is a simplified non-linear single neuron model that has been used e.g. to build large-scale models of the brain (Izhikevich & Edelman, 2008) and to understand oscillatory phenomena in the cortex (Izhikevich, 2003; Domhof & Tiesinga, 2021) and the olfactory bulb (Galán et al., 2006). An attractive property of the IN is that a whole range of different response dynamics can be simulated (Fig. S1) depending on the setting of the parameters  $\theta = [a, b, c, d]$  (Izhikevich, 2004). The IN is described by the following pair of ODEs (Izhikevich, 2003):

$$\begin{aligned}\dot{v}(t, v, u) &= 0.04 \cdot v^2 + 5 \cdot v - u + I_{\text{Stim}}(t), \\ \dot{u}(t, v, u) &= a(b \cdot v - u),\end{aligned}\quad (9)$$

where  $v$  is the membrane potential,  $u$  is a recovery variable and  $I_{\text{Stim}}$  is a given input current. Whenever the spike threshold, a spike is triggered and the neuron is reset in the next time step of the simulation:

$$\begin{aligned}v(t + \Delta t_{\text{Sp}}) &= c, \\ u(t + \Delta t_{\text{Sp}}) &= u(t) + d,\end{aligned}\quad (10)$$

where  $\Delta t_{\text{Sp}} \geq 0$ . Following the original implementation, we set the threshold to 30. Typically,  $\Delta t_{\text{Sp}} = \Delta t$  is used, but to facilitate the comparison between different step-sizes we used  $\Delta t_{\text{Sp}} = 0$  instead. In the original implementation, the reset can only be triggered after a full integration step. So, whenever  $v(t_{i+1}) \geq 30$  after a step from  $t_i$  and  $t_{i+1}$ , the neuron is reset as described above, i.e.  $v(t_{i+1} + \Delta t_{\text{Sp}}) = c$  and  $u(t_{i+1} + \Delta t_{\text{Sp}}) = u(t_{i+1}) + d$ . This is problematic, because it introduces an error of order  $O(\Delta t)$  (Stewart & Bair, 2009), independent of the solver scheme.

Therefore, in addition to this discrete version of resetting the neuron, we implemented a continuous version based on two complementary strategies. First, we adapted Eq. (9) such that whenever  $\dot{v}$  and  $\dot{u}$  would have been evaluated for

$v(t) \geq 30$ —which can only happen for multi-stage methods—the derivatives were evaluated for  $v(t) = 30$  instead. Second, we implemented the strategy suggested by Stewart and Bair (2009): Every step resulting in a reset is split into two intermediate steps, a step until the threshold is reached, and a step after the reset. For this, the spike-time  $t_{\text{spike}}$  during such a step was estimated as described in Sect. 2.4 with a threshold of  $v_{\text{th}} = 30$ . Then, the pre-reset step solution  $\mathbf{x}(t_{\text{spike}})$  was approximated based on the interpolation strategies described in Sect. 2.3. And finally, the post-reset step solution  $\mathbf{x}(t_{i+1})$  was computed by resetting (see Eq. (10)) and integrating  $\mathbf{x}$  from  $t_{\text{spike}}$  to  $t_{i+1}$ .

### 2.5.2 Single Hodgkin–Huxley neurons

Hodgkin–Huxley (HH) models (Hodgkin & Huxley, 1952) are widely used to simulate single and multi-compartment neurons. We study both the classical HH neuron (Hodgkin & Huxley, 1952) and a single compartment HH-like neuron model (Prinz et al., 2003) prominently used to study the stomatogastric ganglion (STG) (Prinz et al., 2004). Both models are described by ODEs that include, among other state variables, the membrane potential  $v(t)$ :

$$\dot{v}(t) = (I_{\text{Stim}}(t) - \sum_i I_i(\mathbf{x})) / C, \quad (11)$$

where  $C$  is the membrane capacitance,  $I_{\text{Stim}}$  is the stimulation current and  $I_i$  are membrane currents. These membrane currents are described by the following equation:

$$I_i(\mathbf{x}) = \bar{g}_i \cdot m_i(\mathbf{x})^{p_i} \cdot h_i(\mathbf{x}) \cdot (v - E_i), \quad (12)$$

where  $E_i$  is the reversal potential of the current,  $\bar{g}_i$  is the maximum channel conductance,  $p_i$  are integer exponents, and  $m_i$  and  $h_i$  are activation and inactivation functions.  $m_i$  and  $h_i$  were modeled by the following differential equations:

$$\begin{aligned}\dot{m}(v) &= (m_{\infty}(v) - m) / \tau_m(v), \\ \dot{h}(v) &= (h_{\infty}(v) - h) / \tau_h(v),\end{aligned}\quad (13)$$

where  $m_{\infty}$ ,  $\tau_m$ ,  $h_{\infty}$ , and  $\tau_h$  are voltage dependent functions defining the channel's kinetics. For non-inactivating channels,  $h_i$  is removed from Eq. (12). In the classical HH model, this amounts to a 4-dimensional ODE (Ermentrout & Terman, 2010). For the STG neuron, which has eight instead of two membrane currents and also implements a model for the intracellular calcium concentration, the ODE is 13-dimensional (Prinz et al., 2003). The respective parametrizations can be found in Sect. 5.3.

We simulated the HH neuron's response to two different input currents  $I_{\text{Stim}}$ , a step and a noisy step stimulus. Both stimuli were 200 ms long, with  $I_{\text{Stim}}(t) = 0$  for  $t < t_{\text{onset}}$  and  $t \geq t_{\text{offset}}$ , where  $t_{\text{onset}} = 10$  ms and  $t_{\text{offset}} = 190$  ms. The amplitude of the step stimulus for  $t_{\text{onset}} \leq t < t_{\text{offset}}$  was  $I_{\text{Stim}}(t) = 0.2$  mA.



The amplitude of the noisy step stimulus were created by drawing 100 values from a uniform distribution between 0.0 mA and 0.4 mA that were spaced equidistantly between  $t_{\text{onset}}$  and  $t_{\text{offset}}$ . These points were interpolated using a cubic spline with endpoints at  $t_{\text{onset}}$  and  $t_{\text{offset}}$ . At the endpoints both  $I_{\text{stim}}$  and its derivative were set to zero. The single STG neuron was simulated for 3 s using a step stimulus starting at  $t_{\text{onset}} = 0.9$  s with an amplitude of  $I_{\text{stim}}(t) = 3$  nA.

### 2.5.3 STG model

The STG neuron model described above was used by Prinz et al. (2004) in a network of three synaptically coupled neurons, ABPD, LP and PY, to study their firing patterns in dependence of the synaptic and neuronal parametrizations. In the model, there are seven synapses connecting the neurons, that are either modeled as slow or fast synapses. The postsynaptic input current  $I_i$  to a neuron is described by:

$$I_i(\mathbf{x}) = \bar{g}_i \cdot s_i(\mathbf{x}) \cdot (v - E_i), \quad (14)$$

where, similarly to Eq. (12),  $E_i$  is the reversal potential of the current,  $\bar{g}_i$  is the synapse's maximum conductance,  $v$  is the membrane potential of the postsynaptic neuron and  $s$  is the activation function of the synapse.  $s$  is described by the following differential equation:

$$\begin{aligned} \dot{s} &= (\bar{s} - s)/\tau_s, \\ \bar{s} &= (1 + \exp((-35 \text{ mV} - v_{\text{pre}})/5 \text{ mV}))^{-1}, \\ \tau_s &= (1 - \bar{s})/f_s, \end{aligned} \quad (15)$$

where  $v_{\text{pre}}$  is the membrane potential of the presynaptic neuron and  $\tau_s$  and  $f_s$  are constants (see Sect. 5.3).

## 2.6 Quantifying numerical uncertainty

### 2.6.1 Reference solutions

None of the aforementioned neuron models has an analytical solution. It is therefore not possible to compare simulations to the true solutions of the respective IVPs. As a substitute, we computed reference solutions using a deterministic RKDP<sub>a</sub> solver with a tolerance of  $\kappa = 1\text{e-}12$  and a maximum step-size dependent on the model investigated (0.01 ms for IN and HH; 0.1 ms for the STG model). To obtain a reference solution at the same time points of a given fixed step-size solution  $X = [\mathbf{x}(t_0), \dots, \mathbf{x}(t_M)]$ , we forced the reference solver to evaluate  $\mathbf{x}(t)$  at least at all time points  $T = [t_0, \dots, t_M]$  of the given solution. For this, in every step in which the adaptive reference solver automatically picked a step-size that would skip any  $t_i$  in  $T$  by taking a too large step-size  $\Delta t_{i-1}$ , the step-size  $\Delta t_{i-1}$  was clipped such that the step was evaluated exactly at  $\mathbf{x}(t_i)$ . All solutions  $\mathbf{x}(t)$  for  $t$  not in  $T$

were dropped before the comparison. To compare adaptive step-size solvers to reference solutions, we also forced these solvers to evaluate time points on a grid  $T = [t_0, \dots, t_M]$  with time points space equidistantly using a distance of 1 ms.

### 2.6.2 Distance metrics

To estimate the uncertainty for a given neuron model and solver, we computed multiple solutions (samples) with the same probabilistic solver to obtain a distribution of solutions. Based on these sample distributions and the respective reference solutions, we evaluated the distributions of sample-sample distances and sample-reference distances using two different distance measures. As a general measure, we computed Mean Absolute Errors (MAEs) between single traces. If not stated otherwise, MAEs were computed on the simulated membrane potentials  $v(t)$ , because this is typically the quantity of interest. For two traces of equal size  $\mathbf{a} = [a_0, \dots, a_M]$  and  $\mathbf{b} = [b_0, \dots, b_M]$  the MAE was defined as:

$$\text{MAE} = \frac{1}{M} \sum_{i=0}^M |a_i - b_i|. \quad (16)$$

For  $n$  samples from a probabilistic solver, we computed the sample-sample distance distribution  $\text{MAE}_{\text{SM}}$  as the  $n$  MAEs between single samples and the mean trace of the other  $n - 1$  samples. Sample-reference distance distributions  $\text{MAE}_{\text{SR}}$  were computed as the  $n$  MAEs between single samples and the reference solution. In some cases, we also computed the distance between the solution of a corresponding deterministic solver to the reference solution, abbreviated as  $\text{MAE}_{\text{DR}}$ .

As a second metric, we computed ‘‘SPIKE-distances’’ between the spike-times of different solutions (Kreuz et al., 2013). The SPIKE-distance is a bounded distance measure between zero and one that quantifies the dissimilarity between two (or more) spike-trains based on the distances between neighboring spikes. Here, we used an open-source python implementation (Mulansky & Kreuz, 2016).

For plotting, we also computed spike density functions (SDFs) of sample distributions as Gaussian kernel density estimates with a bandwidth optimized through grid-search and tenfold cross-validation using the `Scikit-learn` toolbox (Pedregosa et al., 2011).

## 3 Results

In this study, we explored the potential of probabilistic ODE solvers in computational neuroscience. First, we study the effect of numerical uncertainty on simulations of neuron models and qualitatively show that probabilistic solvers can reveal this uncertainty in a way that is easy to interpret.

Second, we provide examples and guidelines where probabilistic solvers can be useful when conducting a new study. Third, we analyze potential drawbacks of probabilistic solvers, such as computational overhead.

### 3.1 Probabilistic solvers can reveal numerical uncertainty in neuron models

To demonstrate the effect of numerical uncertainty on simulations of single neuron models, we first simulated the classical HH neuron with the step stimulus (Fig. 2A). We computed solutions with a deterministic and probabilistic EE solver for a step-size of  $\Delta t = 0.25$  ms. Additionally, we computed a reference solution. We found that the exact spike-times of the deterministic EE solver differed substantially from the reference solution (spike-time difference  $t_{\text{spike}}^{\text{det.}} - t_{\text{spike}}^{\text{ref.}}$  of the first three spikes: 0.6 ms, 2.3 ms, 4.0 ms). The probabilistic solver revealed this numerical uncertainty with spike-times varying substantially between samples (standard deviation (SD) of the spike-time  $t_{\text{spike}}$  for the first three spikes over all 100 samples: 0.2 ms, 0.9 ms, 1.1 ms).

Next we simulated single INs with different parametrizations  $\theta_i$  and response dynamics (Izhikevich, 2004). Using the original step-sizes  $\Delta t$  and input currents  $I_i$ , we compute solutions with the original solver scheme—which is related to a  $\text{FE}_f$  solver (Sect. 5.2)—a deterministic FE scheme and a probabilistic  $\text{FE}_f^t$  solver. We found, that for the “Inhibition-induced spiking” neuron all solvers produced similar spiking patterns in response to a negative current step (Fig. 2B). However, the original solver produced longer intervals between the spikes compared to the reference, resulting in only three instead of four spikes. The deterministic FE solution matched the reference better (e.g. both had four spikes), but the spike-times were still off by several milliseconds (spike-time difference  $t_{\text{spike}}^{\text{det.}} - t_{\text{spike}}^{\text{ref.}}$  of the last two spikes: 8.2 ms, 3.9 ms). The probabilistic solver revealed this numerical uncertainty (SD of the spike-time  $t_{\text{spike}}$  of the first three spikes: 3.0 ms, 5.6 ms, 7.1 ms).

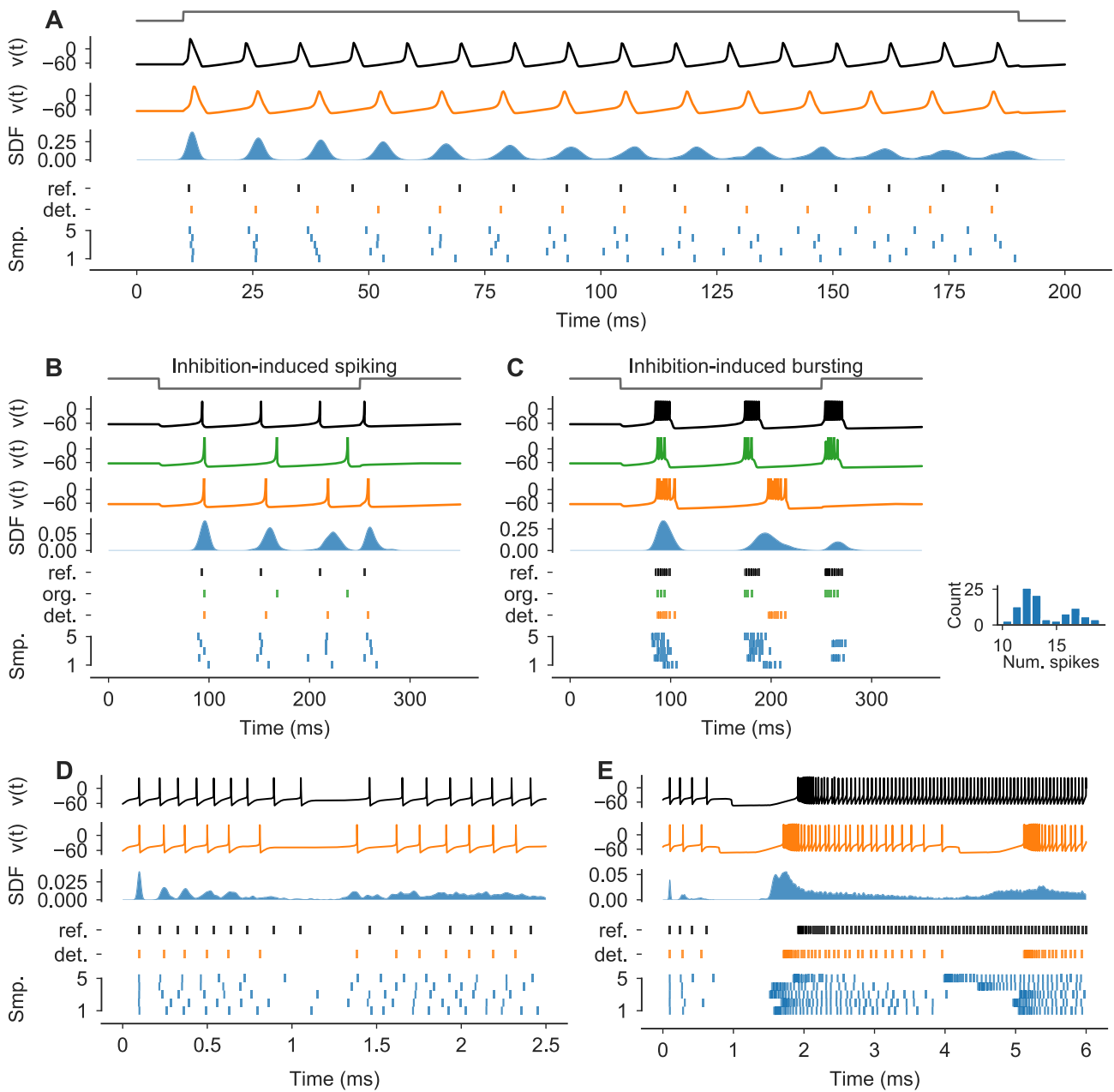
Similarly, for the “Inhibition-induced bursting” neuron the solution from the original solver and the deterministic FE solver were qualitatively broadly consistent with the reference solution (Fig. 2C). In all simulations, the neuron responded with spike bursts to a negative stimulus current step. The spike-times and the number of spikes of the original solution ( $n_{\text{spikes}} = 11$ ) and the deterministic FE solution ( $n_{\text{spikes}} = 14$ ) differed substantially from the reference ( $n_{\text{spikes}} = 33$ ) though, with the FE solution having only two bursts instead of three during the simulated period. Here, the probabilistic solver revealed the substantial uncertainty in the spike-times and number of spikes ( $\bar{n}_{\text{spikes}} = 13.8$  (SD 2.5), where  $\bar{n}$  denotes the sample mean. See also histogram in Fig. 2C), with around 33% of the samples having a third

burst (Fig. 2C, bottom). All 16 simulated parametrizations are shown in Fig. S1.

To provide an example of a neuronal network, we simulated the STG model for two parametrizations (Fig. 2D and E) that only differ in their synaptic conductances (see Sect. 2.5.3). We computed solutions with a reference solver, a deterministic and a probabilistic EE solver. We focused the analysis on the LP neuron for simplicity. For the first parametrization (Fig. 2D), the LP neuron showed continuous spiking in all simulations. Similar to the HH neuron, we found differences in the exact spike-times and number of spikes between the reference ( $n_{\text{spikes}} = 17$ ) and the deterministic EE solution ( $n_{\text{spikes}} = 13$ ). The uncertainty was again revealed by the probabilistic solver ( $\bar{n}_{\text{spikes}} = 14.6$  (SD 1.3), Fig. 2D). The second parametrization resulted in a different spiking behavior of the LP neuron (Fig. 2E). Here, the neuron started to fire at a high frequency for a prolonged time after approximately two seconds. In the reference solution, the neuron continued to fire. In contrast, in the deterministic solution, the neuron stopped after about another two seconds to then start another burst shortly later. While this also happened in all generated samples from the probabilistic solvers, the sample distribution still indicated a high uncertainty about the duration of the firing periods (Fig. 2E). Simulations of all five synaptic parametrizations from the original paper (Prinz et al., 2004) are shown in Fig. S2.

Finally, we turned to a single STG neuron and stimulated the response to a step stimulus (Fig. 3) based on the original publication (Prinz et al., 2003). Here, we compared the numerical uncertainty in two different state variables, namely the voltage  $v(t)$  (Fig. 3A) and the intracellular calcium  $\text{Ca}(t)$  (Fig. 3B). We found that the numerical uncertainty differed strongly between these state variables, and was much higher for  $v(t)$  (Fig. 3). While this is expected because of the transient and brief nature of spikes in contrast to the slower changing calcium, it highlights the power of probabilistic ODE solvers, as they can guide the choice of the solver and step-size parameter dependent on the quantity of interest and the desired accuracy without requiring detailed knowledge about the model and its kinetics.

All the examples in Fig. 2 used first order methods. To also provide an example where higher order solvers with low tolerances yield solutions qualitatively different from the reference solution, we simulated the classical HH neuron’s response for 50 ms to a step stimulus with an amplitude of 0.022406 mA and  $t_{\text{onset}} = 10$  ms and  $t_{\text{offset}} = 40$  ms. This amplitude did not evoke a single spike in the reference solver (Fig. 4A), but was very close to the threshold, i.e. slightly larger amplitudes (e.g. 0.022410 mA) did produce a spike for the reference solver. When simulating this model with a  $\text{RKDP}_a$  solver, we found that for tolerances of  $\kappa = 1e-3$  and  $\kappa = 1e-5$  the solutions did contain a spike (Fig. 4A and



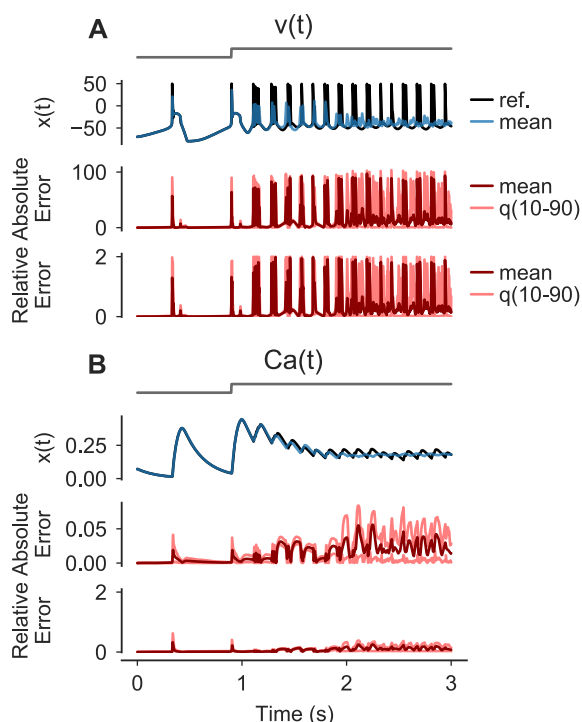
**Fig. 2 Neuron simulations can be subject to substantial numerical uncertainty.** **A** Simulations of the classical HH model for the step stimulus  $I_{stim}$  (normalized stimulus in gray). Solutions for  $v(t)$  are shown for a reference solver (black) and a deterministic EE solver with  $\Delta t = 0.25$  ms (orange) together with a spike density function (SDF) estimated from 100 samples from a probabilistic  $EE_f^t$  solver using the same step-size (blue). *Bottom panel:* Spike-times of the reference (black), the deterministic EE solution (orange) and for 5 randomly selected samples from the probabilistic solver (blue). **B, C** As in A, but for simulations of the IN model for two different parametrizations  $\theta_i$  (see Sect. 5.3) and stimuli  $I_{stim}$  (normalized

stimuli in gray). Here, a  $FE_f^t$  solver and its deterministic version were used. Additionally, a solution was computed using the original solver (green). Based on the original publication, the step-size  $\Delta t$  was set to 0.5 ms for all but the reference solver. For plotting,  $v(t)$  were clipped at 30. In C, the number of spikes of the 100 samples are shown as a histogram. **D, E** As in A, but for simulations of the STG neuron network model for two different synaptic parametrizations  $\theta_{syn}^b$  and  $\theta_{syn}^d$  (see Sect. 5.3), respectively. The membrane potential  $v(t)$  and spike-times are shown for the LP neuron. solutions were computed using an  $EE_f^t$  solver and its deterministic counterpart with a step-size of  $\Delta t = 0.1$  ms

B). Only a tolerance as small as  $\kappa = 1e-7$  yielded a solution with no spike for this solver (Fig. 4C). Simulating the model with probabilistic solvers revealed this numerical

uncertainty for both  $\kappa = 1e-3$  and  $\kappa = 1e-5$ , with a fraction of samples containing one and a fraction containing zero spikes in both cases (Fig. 4D).



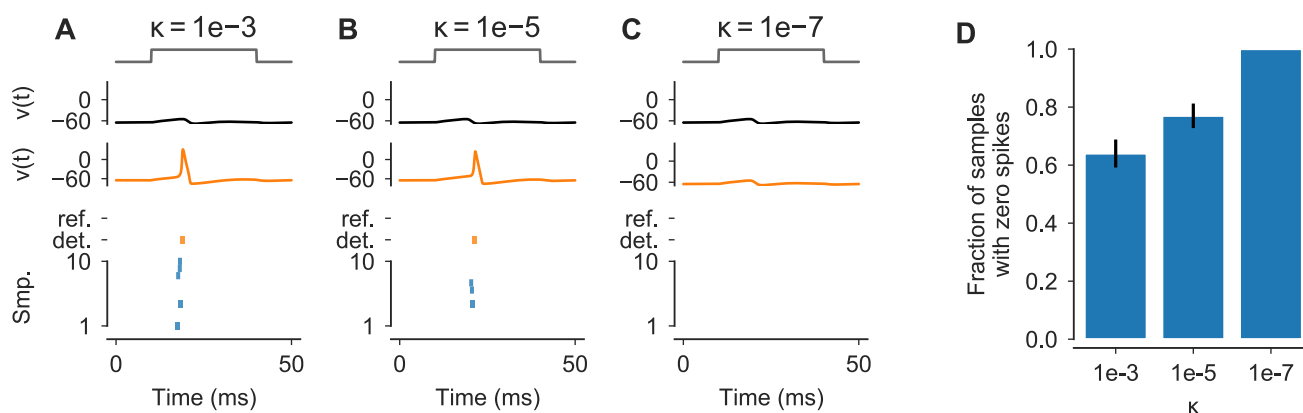


**Fig. 3 Numerical uncertainty can vary between state variables. A, B** Simulations of a single STG neuron in response to a step-stimulus (gray, top row). Solutions were computed using a reference solver and by drawing 100 samples from an  $EE_r^t$  solver with  $\Delta t = 0.1$  ms. The reference solution (black) and the mean over the samples (blue) is shown for two state variables: the membrane potential  $v(t)$  (A, second row) and the intracellular calcium  $Ca(t)$  (B, second row). For both state variables, the absolute error  $AE(t) = |x_{\text{sample}}(t) - x_{\text{ref.}}(t)|$  (third row) and the relative error  $RE(t) = |x_{\text{sample}}(t) - x_{\text{ref.}}(t)| / \max(|x_{\text{sample}}(t)|, |x_{\text{ref.}}(t)|)$  (bottom row) between sample and reference traces are shown as means and the 10th and 90th percentiles over all samples, respectively

### 3.2 Probabilistic solvers can guide solver selection

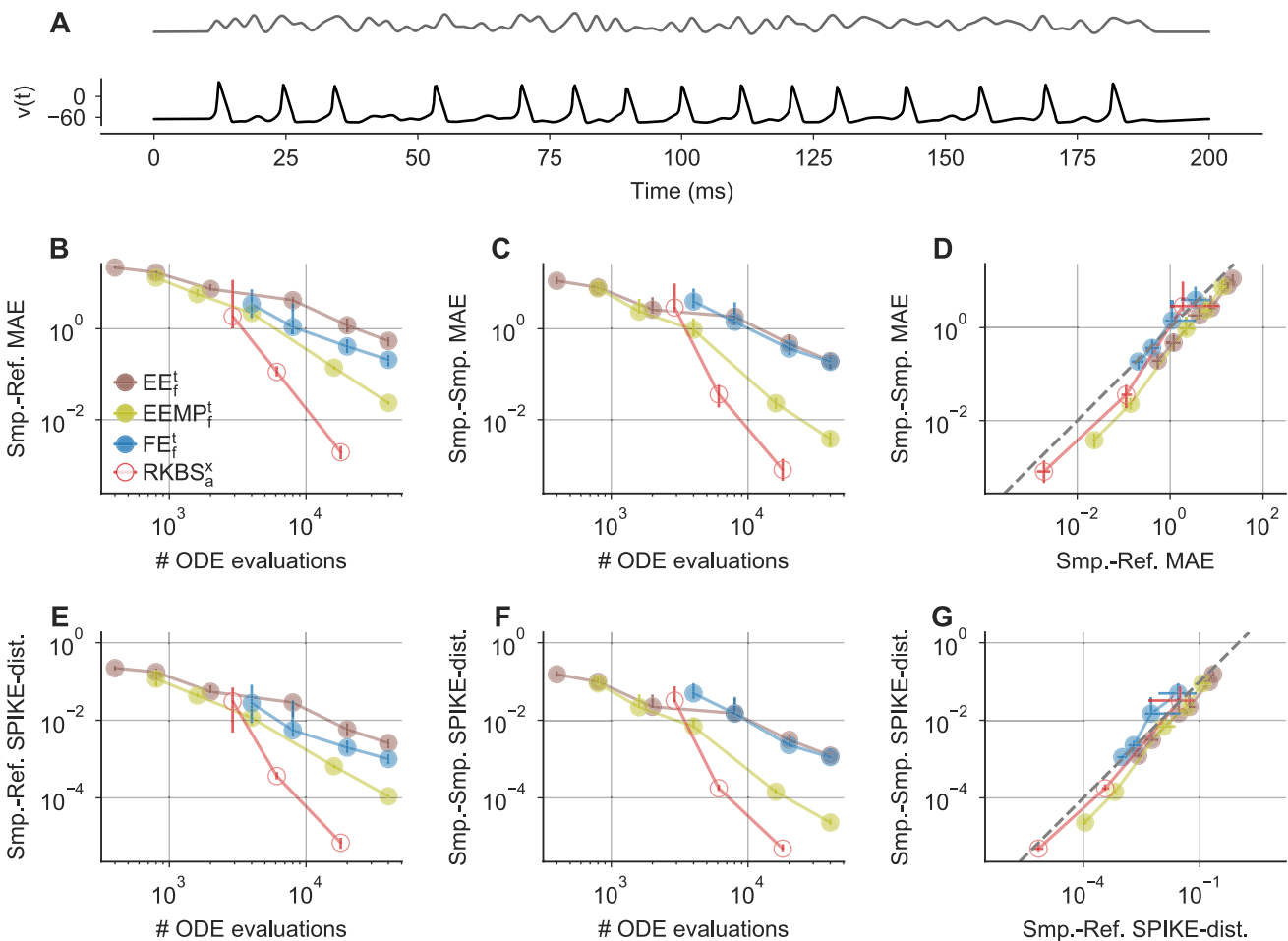
To demonstrate how probabilistic ODE solvers can be used to compare the accuracy vs. run time tradeoff between different solver schemes, we simulated the HH neuron’s response to the noisy step stimulus (Fig. 5A) using the following probabilistic solvers:  $EE_r^t$ ,  $EEMP_f^t$ ,  $FE_r^t$  and  $RKBS_a^x$ . To this end, we computed sample-reference Mean Absolute Errors ( $MAE_{SR}$ ) and sample-reference SPIKE-distances (see Sect. 2.6.2) for each solver as an estimate of the numerical error induced. We compared these errors to the number of ODE evaluations a corresponding deterministic solver would need. We found that the exponential integrators EE and EEMP allowed computing solutions with the fewest ODE evaluations, as they terminated successfully even for the relatively large step-size  $\Delta t = 0.5$  ms (Fig. 5B). In contrast, when using the FE solver, all step-sizes  $\Delta t \gg 0.05$  ms resulted in floating-point overflow errors and therefore in both useless and incomplete solutions. However, when choosing a sufficiently small step-size of  $\Delta t \leq 0.05$  ms the samples obtained with the FE method had on average a smaller error compared to the EE method (Fig. 5B and E). From the methods tested, the adaptive RKBS method was the most efficient one, i.e. it produced the most accurate solutions for the fewest number of ODE solutions, but it also required a substantially higher number of minimum ODE evaluations to successfully terminate compared to the exponential integrators (Fig. 5B and E).

In principle, a very similar analysis could also have been done with deterministic solvers. However, probabilistic solvers have two advantages. First, they yield sample distributions instead of single solutions which make it possible to



**Fig. 4 Numerical uncertainty affects also higher order methods. A–C** Simulations of the classical HH model for a step stimulus  $I_{\text{Stim}}$  with an amplitude of 0.022406 mA (normalized stimulus in gray). Solutions for  $v(t)$  are shown for a reference solver (black) and a deterministic  $RKDP_a$  solver with  $\kappa = 1e-3$ ,  $\kappa = 1e-5$  and  $\kappa = 1e-7$ , respectively (orange). *Bottom panels:* Spike-times of the reference

(black), the deterministic solutions (orange) and for 10 samples from probabilistic  $RKDP_a^x$  solvers with  $\kappa = 1e-3$ ,  $\kappa = 1e-5$  and  $\kappa = 1e-7$ , respectively (blue). **D** Fraction of samples ( $n = 100$ ) from the probabilistic solvers in (A–C) that had no spike, shown as mean and standard error. All other samples had exactly one spike



**Fig. 5 Probabilistic solvers can be used to compare different solver schemes.** **A** Reference solutions of  $v(t)$  (black) for the Hodgkin–Huxley model stimulated with the noisy step stimulus (normalized stimulus in gray). **B** Mean Absolute Errors  $MAE_{SR}$  between sample traces of  $v(t)$  and the respective reference solutions for different solver schemes (legend) and step-sizes / tolerances. Mean Absolute Errors from 100 samples are shown as medians (dots) and 10th to 90th percentiles (vertical

lines) as a function of the number of ODE evaluations of a corresponding deterministic solver (x-axis). **C** As in B, but for sample-sample Mean Absolute Errors  $MAE_{SM}$ . **D** As in B, but for sample-sample Mean Absolute Errors  $MAE_{SM}$  vs. sample-reference Mean Absolute Errors  $MAE_{SR}$ . **E–G** As in B–D, but for using the distance measure SPIKE-distance (see Sect. 2.6.2) instead of Mean Absolute Errors

compute confidence intervals etc. when comparing different solver outputs. Second, and more crucially, probabilistic solvers do not require a reference solution to estimate how numerical errors in a solution affect quantities of interest such as spike-times. For a sufficiently calibrated probabilistic solver, the sample distribution, i.e. the solver’s output, can be used to estimate the numerical error of the solver. In Fig. 5C and F we computed the sample-sample distances which are independent of the reference, for the same samples used in Fig. 5B and E. We found that the mean sample-reference distances were highly similar to the respective mean sample-sample distances for all solvers for both the Mean Absolute Error (Fig. 5B–D) and the SPIKE-distance (Fig. 5E–G). Therefore, the solver comparison described above could have also been based on sample-sample distance instead of the sample-reference distances (e.g.  $MAE_{SM}$

instead of  $MAE_{SR}$ ), and thus would not have required a reference solution.

### 3.3 Calibration of probabilistic solvers

The mean sample-sample distance (e.g. measured as  $MAE_{SM}$ ) is only then a good approximation to the mean sample-reference distance (e.g. measured as  $MAE_{SR}$ ), when the probabilistic solver is well calibrated. Ideally, the magnitude of the perturbation is large enough to capture the numerical uncertainty of the underlying numerical integration, but it is not too large to severely reduce the accuracy of the integration scheme. To quantify the calibration of different solvers, we therefore defined two metrics, the ratio  $R_S = MAE_{SM}/MAE_{SR}$  and the ratio  $R_D = MAE_{DR}/MAE_{SR}$ , where  $MAE_{DR}$  is the distance between a corresponding

deterministic solution and the reference.  $R_S$  is close to zero if the perturbation is too small (i.e. the sample-sample distance is much smaller than the sample-reference distance) and close to one if the perturbation is sufficiently large to not underestimate the numerical uncertainty (i.e. the sample-sample distance can be used as an approximate measure of the sample-reference distance)<sup>1</sup>. However,  $R_S$  is also close to one, when the perturbation is too large and the solver output is dominated by the perturbation rather than the underlying ODE. This is why we also consider  $R_D$  to evaluate the calibration.  $R_D$  is close to one if the perturbation is either too small to affect the model output (i.e. all samples are approximately equal to the deterministic solution) or if samples are on average approximately equally close to the reference than the deterministic solution.  $R_D$  is close to zero, if the perturbation is too large and the perturbation severely reduces the solver accuracy<sup>2</sup>.

For a well calibrated solver,  $R_S$  is close to one, such that the sample-reference distance  $\overline{\text{MAE}}_{\text{SR}}$  can be estimated from sample-sample distance  $\overline{\text{MAE}}_{\text{SM}}$  while  $R_D$  is close to or larger than one, such that the perturbation does not decrease the solver accuracy.

The magnitude of the perturbation can be adjusted with the perturbation parameter  $\sigma$  that we defined for both the state and step-size perturbation (see Sect. 2.1). To analyze how the parameter  $\sigma$  affects the calibration of the perturbation and to test for which  $\sigma$  the solvers are well calibrated, we simulated the classical HH neuron in response to the noisy and the step stimulus with probabilistic solvers for a range of perturbation parameters (Fig. 6). First, we used a probabilistic  $\text{EE}_f^t$  solver and computed  $\overline{\text{MAE}}_{\text{SM}}$ ,  $\overline{\text{MAE}}_{\text{SR}}$  and  $\overline{\text{MAE}}_{\text{DR}}$  and the ratios of the distribution means  $R_S$  and  $R_D$  for different  $\sigma$  ranging from 0.0625 to 16 for the noisy stimulus (Fig. 6A–C). As expected, we found that with increasing  $\sigma$ , the mean sample-sample distance  $\overline{\text{MAE}}_{\text{SM}}$  converged to the mean sample-reference distance  $\overline{\text{MAE}}_{\text{SR}}$ , and for sufficiently large  $\sigma$  the mean sample-sample distance  $\overline{\text{MAE}}_{\text{SM}}$  could therefore be used as an approximate measure of mean sample-reference distance  $\overline{\text{MAE}}_{\text{SR}}$  (Fig. 6A and B). For example, for  $\sigma = 0.25$ , the perturbation magnitude was too small and the solver was underestimating the numerical uncertainty: Here, the mean sample-sample distances was much smaller  $\overline{\text{MAE}}_{\text{SM}}$  (0.33) than the mean sample-reference

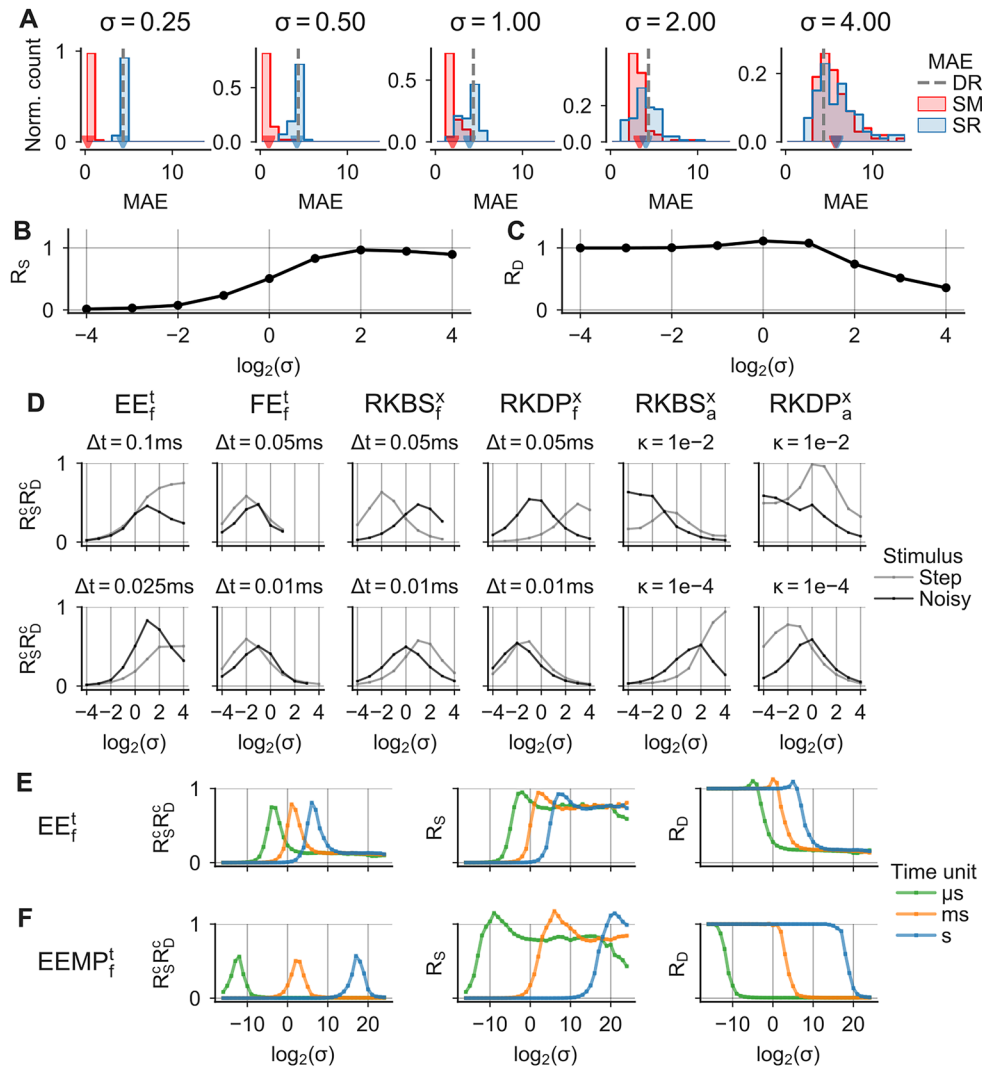
distance  $\overline{\text{MAE}}_{\text{SR}}$  (4.35) (Fig. 6A) with all sample-reference distances distributed narrowly ( $\overline{\text{MAE}}_{\text{SR}}$  10th to 90th percentiles: [4.14, 4.56]) around the deterministic-reference distance  $\overline{\text{MAE}}_{\text{DR}}$  (4.36), indicating that all samples were very close to the deterministic solution, despite the numerical error. When using  $\sigma \geq 4$ , the mean sample-reference distance was higher than the deterministic-reference distance (Fig. 6A and C), indicating a loss of solver accuracy caused by the perturbation (e.g.  $R_D = 0.51$  for  $\sigma = 8$ ). Here, the best calibration was achieved with  $\sigma = 2$ , with distributions of  $\overline{\text{MAE}}_{\text{SM}}$  close to  $\overline{\text{MAE}}_{\text{SR}}$  ( $R_S$ : 0.83; Fig. 6A and B) and with the mean sample accuracy close to the accuracy of the deterministic solution ( $R_D$ : 1.08; Fig. 6A and C).

To provide an overview of the calibration for different solvers settings, we defined a scalar measure for the “goodness of calibration”  $R_S^c R_D^c$ , where  $R_D^c = \min(R_D, 1)$  and  $R_S^c = 1 - |1 - R_S|$ , which is ideally close to one.  $R_S^c R_D^c$  is close to zero for either an underestimation of the numerical uncertainty ( $R_S^c \approx 0$ ) or for a too strong perturbation that renders the solver output useless ( $R_D^c \approx 0$ ). We used  $R_S^c$  and  $R_D^c$  instead of  $R_S$  and  $R_D$ , because in some cases  $R_S$  and  $R_D$  took values larger than one (for example see Fig. S3) which would make their product more difficult to interpret. We simply clipped  $R_D$ , because here we only cared about the samples being at least as close to the reference solution as the deterministic solution. For  $R_S$  on the other hand, values larger than one are not beneficial, as  $R_S$  approximates how well the sample-reference distance can be estimated from the sample distribution alone.  $R_S^c$  therefore measures the deviance of  $R_S$  from one.

We computed  $R_S^c R_D^c$  for different probabilistic solvers and step-sizes—including the  $\text{EE}_f^t$  solver with  $\Delta t = 0.025$  ms used above—for the HH neuron stimulated with the step and noisy step stimulus (Fig. 6D). The respective values for  $R_S$  and  $R_D$  are shown in Fig. S3. We found that using a perturbation parameter of  $\sigma = 1$ , which we used as default in other experiments, produced reasonably calibrated solutions overall. However, in most cases  $\sigma = 1$  was also not ideal. For example for  $\text{EE}_f^t$ , larger values (e.g.  $\sigma = 2$  or  $\sigma = 4$ ) resulted in better calibration, whereas for  $\text{FE}_f^t$  the calibration was improved using smaller values (e.g.  $\sigma = 0.5$  or  $\sigma = 0.25$ ). For the state perturbed Runge–Kutta methods RKBS and RKDP with fixed step-sizes, the best  $\sigma$  were close to one, but it was both step-size and stimulus dependent if slightly larger or smaller values would result in better calibration. The same Runge–Kutta methods with adaptive step-sizes were well calibrated for a wide range of perturbation parameters  $\sigma$ , including very small ones (e.g.  $\sigma = 0.0625$ ), especially in the high tolerance case ( $\kappa = 1e - 2$ ). This is likely because even small perturbations cause the solvers to take different step-sizes and therefore to evaluate the ODE at different time points. When using the state perturbation on a new neuron

<sup>1</sup>  $R_S$  can take values larger than one, for example when the sample distribution is bimodal, the majority of samples is much closer to one of the modes (e.g. a fraction of samples has two and the larger fraction has three spike bursts as in Fig. 2C) and the reference solution is close to the larger mode.

<sup>2</sup> In some cases,  $R_D$  takes values larger than one, which means that the perturbation increases the solver accuracy on average (e.g. see Fig. 2C). This happens, for example, when the deterministic solution is missing a spike but almost reaches the model’s spike threshold, and the perturbation is strong enough to generate the missing spike in some samples.



**Fig. 6 Calibration of probabilistic solvers.** Simulations of the HH model with different perturbation parameters  $\sigma$ . **A** MAE distributions for 100 samples computed with an  $EE_f^t$  solver ( $\Delta t = 0.025$  ms) as  $MAE_{SR}$  (blue),  $MAE_{SM}$  (red) and  $MAE_{DR}$  (grey line). Distribution means are highlighted (triangles). **B**  $R_S = MAE_{SM}/MAE_{SR}$  as a function of  $\sigma$ . **C** As in B, but for  $R_D = MAE_{DR}/MAE_{SR}$ . **D** “Goodness of calibration”  $R_S^c R_D^c$  for different solvers (column titles) and step-sizes

$\Delta t$  / tolerances  $\kappa$  (panel titles) for two stimuli (legend) as a function of  $\sigma$ . **E, F** Simulations using an  $EE_f^t$  and an  $EEMP_f^t$  solver, respectively. The model’s ODEs were simulated in microseconds (green), milliseconds (orange) and seconds (blue), with step-sizes  $\Delta t$  of 25  $\mu s$ , 0.025 ms and 0.000025 s, respectively. MAEs were computed for perturbation parameters  $\sigma$  ranging from  $1e-5$  to  $1e7$ : (left)  $R_S^c R_D^c$ , (middle)  $R_S = MAE_{SM}/MAE_{SR}$ , (right)  $R_D = MAE_{DR}/MAE_{SR}$

model, setting  $\sigma = 1$  may therefore be a good strategy to get a solver with reasonable, though often not ideal, calibration.

For the step-size perturbation on the other hand, setting  $\sigma = 1$  can also result in extremely poor calibration. To illustrate this, we simulated the Hodgkin–Huxley model with different units for the time  $t$ , meaning that we rescaled the values for  $t$  and  $\mathbf{x}$  equally. As expected, this rescaling did not affect the solutions of deterministic or state perturbed solvers. For the step-size perturbation method, however, the step-size  $\Delta t$  is treated as a unit-less quantity during the perturbation step (Eq. (4)) which makes the method sensitive to such unit changes. We simulated the model in milliseconds,

microseconds and seconds for a wide range of perturbation parameters  $\sigma$  and computed the goodness of calibration  $R_S^c R_D^c$  for an  $EE_f^t$  (Fig. 6E) and an  $EEMP_f^t$  solver (Fig. 6F). As expected, the perturbation parameter  $\sigma$  resulting in the best calibration differed by orders of magnitude (Fig. 6E and F). This is, because here the best calibration did always result in the same ratio between the step-size  $\Delta t$  and the standard deviation of the perturbation distribution  $\mathcal{P}$ . Obviously, the step-size perturbation could be made invariant to such unit changes, but this would not address the underlying problem: A priori it is not clear how  $\sigma$ , or more generally the distribution  $\mathcal{P}$ , should be set. Given that the time unit in

which a model is simulated is a relatively arbitrary choice, the reasonable calibration achieved with  $\sigma = 1$  in examples above was at least partially a side effect of all models being simulated in milliseconds with comparable kinetic and is not a generally applicable rule.

Therefore, especially in the case of step-size perturbation, it is advisable to calibrate the perturbation before simulating a new model. In many practical applications, such as simulation-based parameter inference for neuron models (Oesterle et al., 2020), models are simulated repeatedly with slight modifications in their parametrization. Here, it would be impractical to recalibrate the solvers for every evaluation, but on the other hand, it would cause only little overhead to calibrate the perturbation once in the beginning. We therefore tested how transferable the calibration for one model parametrization is to a wider range of other parametrizations. First, we calibrated an  $EE_f^1$  solver by maximizing the goodness of calibration  $R_S^c R_D^c$  through a grid-search over  $\sigma$  (Fig. S4A) on a HH neuron model using the default parameters for the conductances (Sect. 5.3) and a low frequency noisy step stimulus (Fig. S4). Using the optimized perturbation parameter  $\sigma$ , we computed the goodness of calibration for 100 different neuron parameter sets (Fig. S4B) which resulted in a variety of spiking patterns (Fig. S4C and D). For the vast majority of parametrizations, the perturbation was well calibrated (Fig. S4E): For almost half the parametrizations (39 out of 100)  $R_S^c R_D^c$  was even higher than for the original parametrization and in all cases it was larger than 0.3.

### 3.4 Computational overhead

Probabilistic solvers based on state or step-size perturbation increase the computational costs for two reasons. First, they are sampling based and require computing multiple solutions for a single IVP. While this process can be parallelized, it nevertheless comes with a computational overhead, especially if it conflicts with other computations using parallelized model evaluation, e.g. in simulation-based inference where the same model is evaluated for different model parameters (Cranmer et al., 2020; Gonçalves et al., 2020). Second, probabilistic solvers induce a computational overhead per solution computed relative to their deterministic counterparts. We analyzed both aspects in the following.

#### 3.4.1 Required number of samples

To empirically determine the number of samples necessary to obtain a reliable measure of numerical uncertainty, we simulated the classical HH neuron with probabilistic solvers for the step and noisy step stimulus. To this end, we computed mean sample-sample distances  $\overline{MAE}_{SM-n}$  for small numbers of samples  $n$ , and divided them by the mean

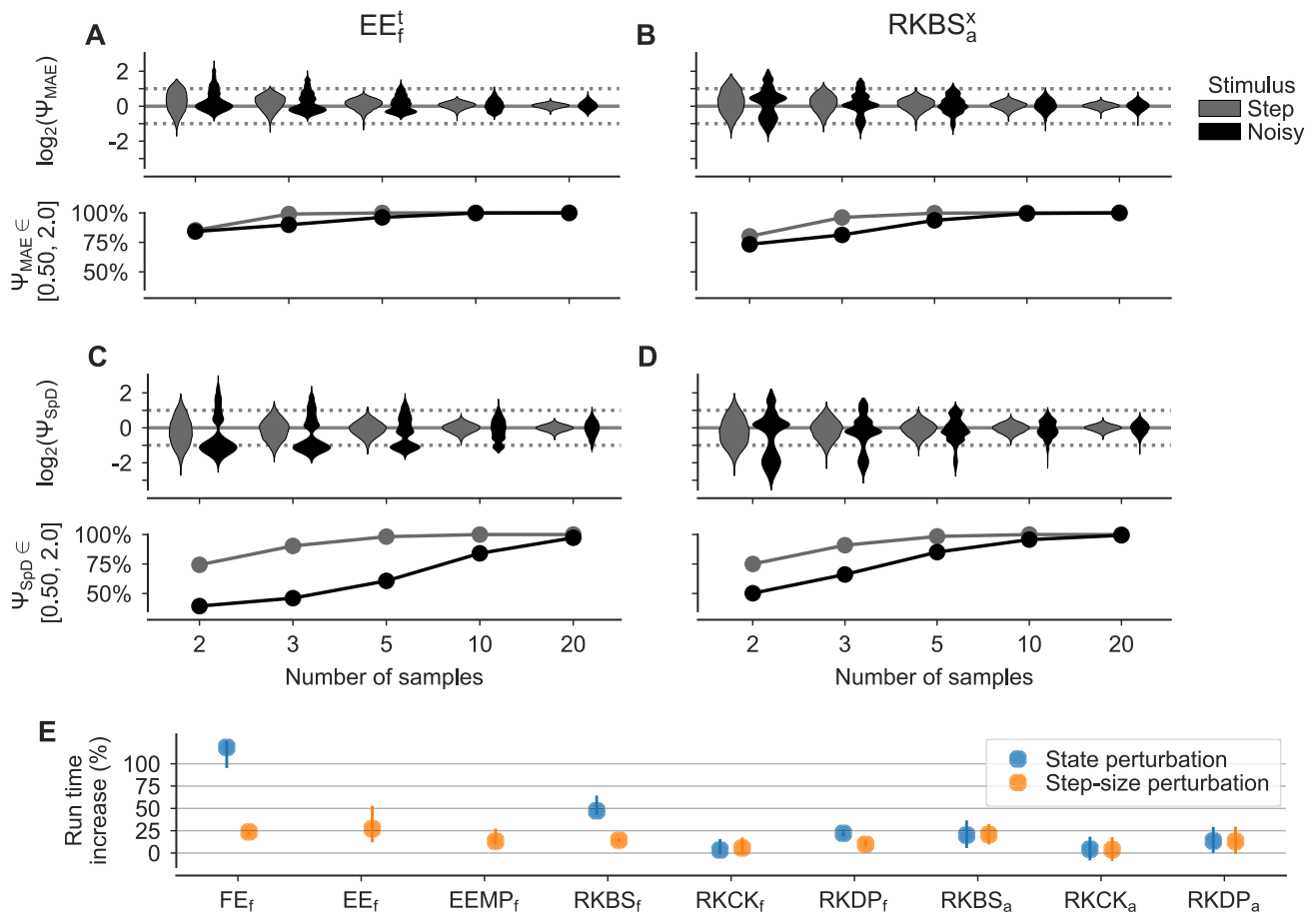
sample-sample distances for a much larger number of samples (300) to obtain normalized sample-sample distances  $\Psi_{MAE}(n) = \overline{MAE}_{SM-n} / \overline{MAE}_{SM-300}$ . Similarly, we computed normalized sample-sample SPIKE distances  $\Psi_{SPD}$  for the spike times. We bootstrapped distributions of  $\Psi_{MAE}$  (Fig. 7A and B) and  $\Psi_{SPD}$  (Fig. 7C and D) to see how likely a certain number of samples  $n$  would result in a good estimate of the true sample-sample distance. We found that two samples were often already sufficient to estimate the sample-sample MAE. For example, for the step stimulus and  $n = 2$ , more than 80% of the bootstrapped  $\Psi_{MAE}$  were in  $[0.5, 2.0]$  with little difference between the solvers  $EE_f^1$  (Fig. 7A) and  $RKBS_a^x$  (Fig. 7B). Reliably estimating the sample-sample SPIKE-distance required more samples, especially for the noisy stimulus. Still, only 10 samples were sufficient to obtain a  $\Psi_{SPD}$  in  $[0.5, 2.0]$  with high probability ( $> 80\%$ ) for both solvers and stimuli.

#### 3.4.2 Overhead per sample

In addition to the computational overhead caused by the computation of multiple samples, probabilistic methods also come with a computational overhead per solution. For the state perturbation this overhead has three components. First, one needs to compute the local error estimator, which only causes overhead for fixed step-sizes since for adaptive methods the local error estimator needs to be computed anyway. The second potential source of overhead is that the “First Same As Last” property—i.e. that the last stage in one step can be used as the first stage of the next step, which is used in RKBS and RKDP—is not applicable. This is because the last stage is computed before the perturbation, and after the perturbation the evaluation of the ODE is not valid anymore. Lastly, the perturbation itself, which includes sampling from a Gaussian, needs to be computed.

In total, this overhead is relatively small for higher order methods optimized for step-size adaptation like RKBS, RKCK and RKDP. For example, the state perturbation for  $RKDP_a$  increases the number of ODE evaluations per step from six to seven (+16%) due to the loss of the First Same As Last property, and for  $RKCK_a$ —which does not make use of this property—no additional ODE evaluation is required. However, for first order methods like FE this overhead severely reduces the computational efficiency because instead of a single ODE evaluation per step, a state perturbed version needs two (+100%). Additionally, lower order methods typically require more steps in total compared to higher order methods, because they are typically used in combination with smaller step-sizes. This increases the total computational costs of the perturbation itself, which is done once per step. For the step-size perturbation, the overhead is reduced to the perturbation and,





**Fig. 7 The computational overhead of probabilistic solver is moderate.** **A, B** Bootstrapped distributions of normalized sample-sample distances  $\Psi_{MAE} = MAE_{SM-n} / MAE_{SM-300}$  as a function of the numbers of samples  $n$ .  $\Psi_{MAE}$  were computed for the classical HH neuron simulated with an  $EE_f^t$  with  $\Delta t = 0.1$  ms (A) and a  $RKBS_a^x$  solver with  $\kappa = 1e-2$  (B) for the step (grey) and noisy step (black) stimulus, respectively. *Top*: Distributions over  $\Psi_{MAE}$  computed by 10000 times repeated random sub-sampling of  $MAE_{SM-n}$  from all 300 generated samples. *Bottom*: Percentages of the bootstrapped  $\Psi_{MAE}$  distributions in  $[0.5, 2.0]$ . **C, D** As in A, B, but for  $\Psi_{SpD} = SpD_{SS-n} / SpD_{SS-300}$ , where  $SpD_{SS-n}$  is the mean of the pairwise SPIKE-distance distribu-

tion. **E** Relative run times for different solver schemes measured for the HH neuron with the noisy step stimulus. For every solver, 100 samples were simulated for both a probabilistic and a corresponding deterministic solver. Relative run times were computed by dividing the run times of the probabilistic samples by the run times of the respective deterministic samples. Distributions were computed by bootstrapping 10000 ratios and are shown as medians and the 10th to 90th percentiles. The step-size of fixed step-size methods was  $\Delta t = 0.01$  ms and the tolerance of adaptive methods was  $\kappa = 1e-4$ . For EE and EEMP, only the step-size perturbation was used

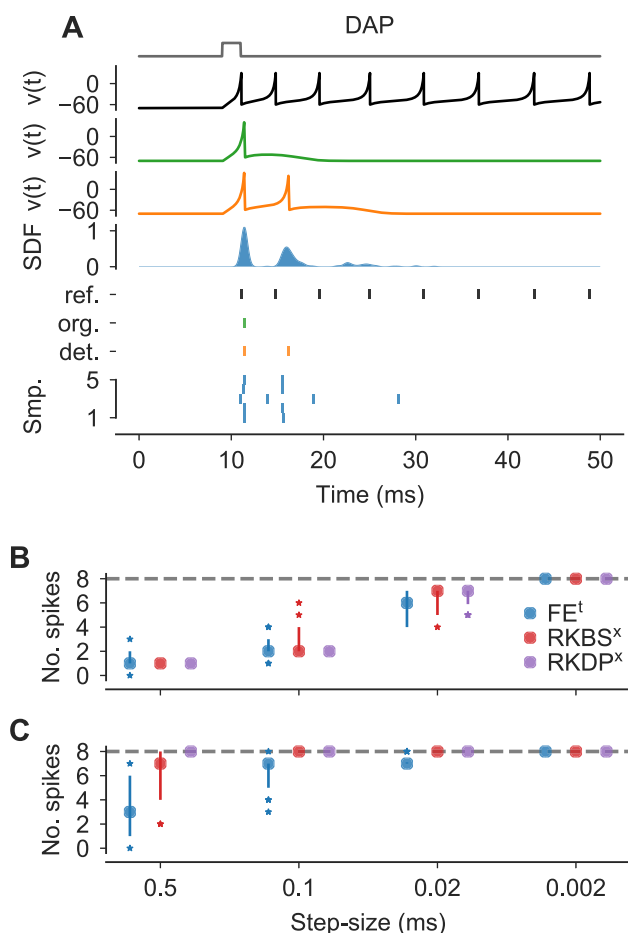
for adaptive step-size methods, the loss of the First Same As Last property.

To quantify this overhead empirically, we simulated the HH neuron with different probabilistic solvers and their deterministic counterparts and compared the run times relative to each other. As expected, for the state perturbation, the computational overhead was larger for the lower order methods (Fig. 7E; on average for  $FE_f^x$ : +113%,  $RKBS_f^x$ : +50%,  $RKCK_f^x$ : +6%,  $RKDP_f^x$ : +23%). The adaptive methods—where the local error estimates were computed not only for the probabilistic, but also for the deterministic methods—showed the smallest increase in run times (+14% on average across all adaptive methods), with  $RKCK_a^x$ , not using the First Same As Last property, having the least overhead

(+5%). For the step-size perturbation, the increase in run times was on average smaller (+16% on average across all methods) and without large differences between the solver schemes and the usage of adaptive or fixed step-sizes.

### 3.5 Errors beyond numerical integration

Finally, we turned back to the “DAP” IN model (Fig. S1), to illustrate numerical uncertainties that cannot be fully captured by the perturbation methods because they are not caused by the numerical approximation of the integral in Eq. (2). For this neuron model we had found a large difference in the number of spikes for the fixed step-size methods, like the original solver, compared to the reference solution



**Fig. 8 Probabilistic solver only account for errors in numerical integration.** **A** Simulations of the “DAP” IN model with a pulse stimulus  $I_{Stim}$  (normalized stimulus in gray). Solutions for  $v(t)$  are shown for a reference solver (black), the original solver scheme (green) and a deterministic FE solver (orange). Based on the original publication, the step-size  $\Delta t$  was set to 0.1 ms. For plotting,  $v(t)$  were clipped at 30. **Bottom panel:** Spike-times are shown for the reference (black), the original solver solution (green), the deterministic FE solution (orange) and for 100 samples from a probabilistic  $FE_f^t$  solver (blue). Samples were sorted by the number of spikes. **B, C** Number of spikes dependent on the step-size for different solver methods (legend) for fixed **A** and pseudo-fixed **B** step-sizes, respectively. Number of spikes are shown for 40 samples as medians, 10th to 90th percentiles (vertical lines) and outliers (stars). The dashed horizontal line refers to the reference solution

(Fig. 8A). While the reference solution had eight spikes during the simulated period, the original solution had only one and a deterministic FE solver had two. While the probabilistic solver  $FE_f^t$  arguably indicated some numerical uncertainty ( $\bar{n}_{spikes} = 2.3$  (SD 0.6)), the number of spikes was still much lower compared to the reference. To better understand the source of this numerical uncertainty, we simulated the “DAP” neuron model with different probabilistic solvers,  $FE_f^t$ ,  $RKBS_f^x$  and  $RKDP_f^x$ .

First, we simulated the neuron for different fixed step-sizes. We found that all probabilistic solvers underestimated the true number of spikes when using relatively large fixed step-sizes (Fig. 6B). For the largest step-size tested,  $\Delta t = 0.5$  ms, only the  $FE_f^t$  solver indicated uncertainty in the number of spikes, whereas for  $RKBS_f^x$  and  $RKDP_f^x$  all samples had only a single spike. When using smaller step-sizes, the probabilistic solvers’ outputs were more indicative of the numerical uncertainty. For  $\Delta t = 0.02$  ms, all solvers produced outputs that were closer ( $\bar{n}_{spikes} = 5.9$  for  $FE$ ,  $\bar{n}_{spikes} = 6.5$  for  $RKBS$  and  $\bar{n}_{spikes} = 6.7$  for  $RKDP$ ) to the reference solution and all methods indicated uncertainty in the number of spikes. With the very small step-size  $\Delta t = 0.002$  ms, all samples from all solvers showed the same number of spikes as the reference and the probabilistic solvers indicated no remaining uncertainty about the number of spikes here.

While these results may be unsatisfactory at first glance, they are not necessarily unexpected. The probabilistic solvers used here can only capture the uncertainty arising through the numerical integration; they cannot capture the error that is introduced by restricting spikes to occur only on a fixed time grid, which is the case for the fixed step-size solvers. We therefore simulated the neuron for the same solvers and step-sizes again, but allowed the solver to take intermediate steps (see Eq. (10)) every time a reset occurred. When using these “pseudo-fixed” step-sizes, we found that  $RKDP_f^x$  still did not indicate uncertainty in the number of spikes for any step-size tested, but now all samples had the same number of spikes as the reference (Fig. 8C). And while  $FE_f^t$  and  $RKBS_f^x$  still underestimated the number of spikes for larger step-sizes on average (e.g. for  $\Delta t = 0.5$  ms:  $\bar{n}_{spikes} = 3.5$  for  $FE$  and  $\bar{n}_{spikes} = 6.3$  for  $RKBS$ ), both indicated high numerical uncertainty (e.g. for  $\Delta t = 0.5$  ms:  $q_{90}(n_{spikes}) = 6$  for  $FE$  and  $q_{90}(n_{spikes}) = 8$  for  $RKBS$ , where  $q_{90}$  is the 90th percentile).

## 4 Discussion

The outcome of neuron simulations is affected by numerical uncertainty arising from the inevitably finite step-sizes used in numerical ODE integration. With standard solvers there is no straightforward way to quantify how this uncertainty affects quantities of interest such as spike-times and the number of spikes.

In this study, we demonstrated how probabilistic solvers can be used to quantify and reveal numerical uncertainty in commonly used neuron models. Crucially, these solvers can be easily implemented and do not require a detailed understanding of the underlying kinetics of the neuron model of interest.

Further, we showed that numerical uncertainty can affect the precise timing and the number of spikes in simulations of neuron models commonly used in neuroscience. We also found that some models and parametrizations are more susceptible to numerical uncertainty than others, and that some solvers employed in the neuroscience literature yield rather large uncertainties. These findings highlight the need for a thorough quantification of numerical uncertainty in neuroscience simulations to strike an informed balance between simulation time and tolerated uncertainty.

The idea to quantify the accuracy or numerical errors of different solvers for mechanistic models in neuroscience is not new. For example, Butera and McCarthy (2004) showed that for small step-sizes, the forward Euler method produces more accurate solutions than the exponential Euler method, which is in agreement with our findings. Börgers and Nectow (2013) on the other hand argued that for Hodgkin–Huxley-like systems exponential integrators—such as exponential Euler and the exponential midpoint Euler—are often the best choice, as they allow for much larger step-sizes especially when high accuracy is not necessary, which is again what we observed. Stewart and Bair (2009) argued in favor of the Parker–Sochacki integration method and showed that it can be used to generate highly accurate solutions for both the Izhikevich and Hodgkin–Huxley model. However, this method has the disadvantage that the ODE system at hand has to be put into the proper form and therefore requires specific knowledge about the model and solver. In a more recent study, Chen et al. (2020) recommended to use splitting methods, such as second-order Strang splitting, instead of exponential integrators.

In contrast to these studies, probabilistic solvers offer a more general approach to tackle the problem of numerical uncertainty. Instead of finding the “best” solver for a specific problem, they produce an easy-to-interpret uncertainty measure that can be analyzed without specific knowledge about the solver or solved neuron model. This allows to easily assess if a solver is sufficiently accurate for a given research question. It can therefore facilitate both the choice of the solver and choice of solver settings such as the step-size.

In this study, we used two simple probabilistic solvers that build on deterministic solver and stochastically perturb the numerical integration. For both, the state (Conrad et al., 2017) and the step-size perturbation (Abdulle & Garegnani, 2020) method, it is crucial that the perturbation is of the right order to neither underestimate the numerical uncertainty nor to reduce the solver accuracy unnecessarily. To be able to adjust the perturbation we introduced a perturbation parameter for both the state and step-size perturbation. We found that using the default value for this parameter yielded good calibration for most of the models we simulated, but slight adjustments often improved the calibration further. However, especially for the step-size perturbation, the calibration can be strongly

dependent on the model and a certain level of calibration by means of adapting the perturbation distribution to the problem is always recommended. Fortunately, calibration may often be transferable between models with similar kinetics, for example in simulation-based inference where models are repeatedly evaluated for different model parametrizations.

A downside of the step-size perturbation is, that it requires a local error estimator to be calibrated. This can introduce a relatively large computational overhead per solution for lower order methods like FE. The step-size perturbation may therefore a better choice for lower order methods. However, for higher order methods like RKDP this difference vanishes and both approaches require an equally small computational overhead per solution. Another advantage of the step-size perturbation is that it preserves desirable properties of the underlying solver schemes (Abdulle & Garegnani, 2020). For example, when Hodgkin–Huxley-like models are solved with exponential integrators like EE or EEMP, the state variables of the activation and inactivation cannot leave their domain  $[0, 1]$  by design of the solvers, a property preserved by the step-size but not the state perturbation.

Beyond the two perturbation methods used and discussed in this study, there are other classes of probabilistic numerical algorithms applicable to the problem studied herein. For instance, Teymur et al. (2021) recently proposed a method that builds on Richardson’s deferred approach to the limit and replaces the deterministic interpolant with a stochastic process. Similar to the perturbation methods, this approach can be used to create probabilistic ODE solvers from established deterministic ones. Another class of probabilistic ODE solvers is constructed using techniques from (nonlinear) Gaussian filtering and smoothing (Schober et al., 2019; Kersting et al., 2019; Tronarp et al., 2019). These methods have the advantage that instead of repeatedly integrating the initial value problem, they only require a single forward integration and return local uncertainty estimates that are proportional to the local truncation error. The disadvantage of Gaussian ODE filters and smoothers is that the uncertainty estimates are Gaussian. This restriction can be lifted by replacing Gaussian filters and smoothers with particle filters and smoothers (Tronarp et al., 2019). These and other methods may further extend the applicability of probabilistic numerics in computational neuroscience. In particular for large neural network simulations, more efficient methods like the filtering approaches will be key in quantifying uncertainty. However, because these methods are not sampling based, their success for simulating neurons will depend on how well they can capture the numerical uncertainty arising from the all-or-none behavior of spikes.

Another aspect that will be crucial for the success of probabilistic numerics in computational neuroscience is to develop and test implicit probabilistic solvers for neuron models. For example, the ODEs of multi-compartment

neuron models are typically stiff which makes implicit solvers the better choice for such models (Mascagni et al., 1989). A priori, it is often not easy to judge whether a ODE system is stiff or not. A noteworthy attempt to tackle this problem is the algorithm by Blundell et al. (2018) that automatically determines whether an implicit or an explicit solver should be used.

## 5 Appendix

### 5.1 Local error estimation and step-size adaptation

To compute the local error estimator  $\epsilon_t$  for a single integration step, step solutions were computed with two different numerical methods that were run in parallel to provide two solutions  $\mathbf{x}_a(t + \Delta t)$  and  $\mathbf{x}_b(t + \Delta t)$  for every step, given  $t$ ,  $\Delta t$  and  $\mathbf{x}(t)$  (see, for example, Dormand and Prince (1980)). The pairs of numerical methods used in this study are listed in Table 1 and described in more detail in Sect. 5.2. In every step, the local error estimator was computed as:

$$\epsilon_t = |\mathbf{x}_a(t + \Delta t) - \mathbf{x}_b(t + \Delta t)|. \quad (17)$$

For adaptive step-size methods, the error estimator  $\epsilon_t = [\epsilon_t^1, \dots, \epsilon_t^d]^\top$  was used to compute an error norm  $\|\mathbf{e}\|$  on  $\mathbf{e} = [e_1, \dots, e_d]^\top$ , where  $d$  was the dimension of the state vector  $\mathbf{x}(t) = [x_1(t), \dots, x_d(t)]^\top$ . For every state variable  $x_i$ ,  $e_i$  was computed as:

$$e_i = \frac{\epsilon_t^i}{\kappa_a + \kappa_r \cdot \max(|x_i(t)|, |x_i(t + \Delta t)|)}, \quad (18)$$

with  $\kappa_a$  and  $\kappa_r$  being the absolute and relative tolerance. For simplicity, we used  $\kappa_a = \kappa_r$  in all simulations and therefore refer to these parameters as the *tolerance*  $\kappa$ .

$\|\mathbf{e}\|$  was computed as the root-mean-square of  $\mathbf{e}$ , i.e.:

$$\|\mathbf{e}\| = \sqrt{\frac{1}{d} \sum_i e_i^2}. \quad (19)$$

If  $\|\mathbf{e}\| < 1$ , the step was accepted, and rejected otherwise. In both cases, the step-size was adapted and the next step-size  $\Delta t_{\text{next}}$  was computed as:

$$\Delta t_{\text{next}} = 0.9 \cdot \Delta t \cdot \min(\max(\|\mathbf{e}\|^{-1/k_{\text{exp}}}, k_{\text{min}}), k_{\text{max}}), \quad (20)$$

where  $k_{\text{min}}$  and  $k_{\text{max}}$  are the minimum and maximum allowed change factors, that we set to typical values of 0.1 and 5 respectively (Hairer et al., 1993).  $k_{\text{exp}}$  was 3 for RKBS, 4 for RKCK, and 5 for RKDP, corresponding to the order of the method. Furthermore, we limited the step-sizes to be always smaller or equal to a maximum step-size  $\Delta t_{\text{max}}$ , which we set to  $\Delta t_{\text{max}} = 1$  ms for all simulations.

### 5.2 Solver details

Runge–Kutta steps were implemented based on the `scipy` implementation (Virtanen et al., 2020). The Butcher tableau for the RKCK method was taken from (Cash & Karp, 1990). Heun's method was used as an error estimator for the FE method and implemented as follows. Given  $t$ ,  $\Delta t$ ,  $\mathbf{x}(t)$ ,  $f(t, \mathbf{x}(t))$  and the deterministic FE solution  $\mathbf{x}_{\text{det}}^{\text{FE}}(t + \Delta t)$  the solution for Heun's method was computed as:

$$\mathbf{x}(t + \Delta) = \mathbf{x}(t) + \Delta t [f(t, \mathbf{x}(t)) + f(t, \mathbf{x}_{\text{det}}^{\text{FE}}(t + \Delta t))]/2 \quad (21)$$

To use the exponential integrators EE and EEMP, the ODEs were cast into the following form:

$$\dot{z}(t, \mathbf{x}(t)) = [z_\infty(\mathbf{x}) - z]/z_\tau(\mathbf{x}), \quad (22)$$

where  $z$  is a state variable of  $\mathbf{x}$  (e.g. the membrane potential) and  $z_\infty(\mathbf{x})$  and  $z_\tau(\mathbf{x})$  are functions depending on  $\mathbf{x}$  but not explicitly on  $t$ . For a derivation of these functions for the HH model see for example Dayan and Abbott (2001). The EE step was implemented as:

$$z(t + \Delta t) = z_\infty(\mathbf{x}) + [z(t) + z_\infty(\mathbf{x}) \exp(-\Delta t/z_\tau(\mathbf{x}))]. \quad (23)$$

The second order exponential integrator EEMP proposed by Börgers and Nectow (2013) builds on the EE method. Given  $t$ ,  $\Delta t$ ,  $\mathbf{x}(t)$ , the half-step EE solution  $\tilde{\mathbf{x}} = \mathbf{x}_{\text{det}}^{\text{EE}}(t + \Delta t/2)$  and the evaluations of  $z_\infty(\tilde{\mathbf{x}})$  and  $z_\tau(\tilde{\mathbf{x}})$  at the half-step, the solution for  $z$  using the EEMP method was computed as:

$$z(t + \Delta t) = z_\infty(\tilde{\mathbf{x}}) + [z(t) + z_\infty(\tilde{\mathbf{x}}) \exp(-\Delta t/z_\tau(\tilde{\mathbf{x}}))]. \quad (24)$$

We also implemented the solver used in the original implementation of the IN neurons, where the IVP was solved with a method similar to FE of fixed step-size  $\Delta t$  (Izhikevich, 2004). The implementation differs from a standard FE scheme in so far, as  $v$  and  $u$  are updated subsequently:

$$\begin{aligned} v(t + \Delta t) &= v(t) + \Delta t \cdot \dot{v}(t, u(t), v(t)), \\ u(t + \Delta t) &= u(t) + \Delta t \cdot \dot{u}(t, u(t), v(t + \Delta t)). \end{aligned} \quad (25)$$

### 5.3 Neuron model parameters

The neuron models simulated in this study were parametrized as follows. The parameters  $\theta = [a, b, c, d]$  and  $I_{\text{stim}}$  of the IN model and the respective original step-sizes  $\Delta t$  were taken from <https://www.izhikevich.org/publications/figure1.m> (Izhikevich, 2004).

The three maximum conductances for the classical HH neuron (see Eq. (12)) were set to  $\bar{g}_{\text{Na}} = 1.2$  mS,  $\bar{g}_{\text{K}} = 0.36$  mS and  $\bar{g}_{\text{leak}} = 0.003$  mS. The membrane capacitance was set to  $C = 0.01$   $\mu\text{F}$  (see Eq. (11)). For all STG neurons, we set the membrane area to  $A = 0.628e - 3$   $\text{cm}^2$  and the membrane capacitance to  $C = A \cdot 1 \mu\text{F}/\text{cm}^2$  (see Eq. (11)). An STG

neuron has eight maximum channel conductances (see Eq. (12)):

$$\theta_{\text{STG-neuron}} = [\bar{g}_{\text{Na}}, \bar{g}_{\text{CaT}}, \bar{g}_{\text{CaS}}, \bar{g}_{\text{A}}, \bar{g}_{\text{KCa}}, \bar{g}_{\text{Kd}}, \bar{g}_{\text{H}}, \bar{g}_{\text{leak}}]. \quad (26)$$

For the single STG neuron, we set  $\theta_{\text{STG-neuron}} = A \cdot [400, 2.5, 10, 50, 20, 0, 0.04, 0]$  mS/cm<sup>2</sup>, taken from an example in Prinz et al. (2003). The STG neuronal network consists of three neuron models ABPD, LP and PY. The network is parametrized by the three neurons' conductances:

$$\begin{aligned} \theta_{\text{ABPD}} &= A \cdot [100, 2.5, 6, 50, 5, 100, 0.01, 0.0] \text{ mS/cm}^2, \\ \theta_{\text{LP}} &= A \cdot [100, 0.0, 4, 20, 0, 25, 0.05, 0.03] \text{ mS/cm}^2, \\ \theta_{\text{PY}} &= A \cdot [100, 2.5, 2, 50, 0, 125, 0.05, 0.01] \text{ mS/cm}^2, \end{aligned} \quad (27)$$

where  $A = 0.628e - 3 \text{ cm}^2$  and the synaptic conductances  $\theta_{\text{syn}}$ :

$$\theta_{\text{syn}} = [\bar{g}_{\text{ABPD-LP}}^{\text{fast}}, \bar{g}_{\text{ABPD-LP}}^{\text{slow}}, \bar{g}_{\text{ABPD-PY}}^{\text{fast}}, \bar{g}_{\text{ABPD-PY}}^{\text{slow}}, \bar{g}_{\text{LP-ABPD}}^{\text{fast}}, \bar{g}_{\text{LP-ABPD}}^{\text{slow}}, \bar{g}_{\text{LP-PY}}^{\text{fast}}, \bar{g}_{\text{LP-PY}}^{\text{slow}}], \quad (28)$$

where for example  $\bar{g}_{\text{ABPD-LP}}^{\text{fast}}$  is the maximum conductance of the fast synapse connecting neuron ABPD (presynaptic) to neuron LP (postsynaptic). We simulated the network for five different synaptic parametrizations taken from the original publication (Prinz et al., 2004):

$$\begin{aligned} \theta_{\text{syn}}^{\text{a}} &= [10, 100, 10, 3, 30, 1, 3] \text{ nS}, \\ \theta_{\text{syn}}^{\text{b}} &= [3, 0, 0, 30, 3, 3, 0] \text{ nS}, \\ \theta_{\text{syn}}^{\text{c}} &= [100, 0, 30, 1, 0, 3, 0] \text{ nS}, \\ \theta_{\text{syn}}^{\text{d}} &= [3, 100, 10, 1, 10, 3, 10] \text{ nS}, \\ \theta_{\text{syn}}^{\text{e}} &= [30, 30, 10, 3, 30, 1, 30] \text{ nS}. \end{aligned} \quad (29)$$

The frequencies  $f_s$  (Eq. (15)) for the fast and slow synapses were 25 Hz and 10 Hz, and the reversal potentials  $E_i$  (Eq. (14)) were -70 mV and -80 mV, respectively (Prinz et al., 2004).

**Supplementary Information** The online version contains supplementary material available at <https://doi.org/10.1007/s10827-022-00827-7>.

**Acknowledgements** This research was funded by the Deutsche Forschungsgemeinschaft through a Heisenberg Professorship (BE5601/8-1, PB), the Excellence Cluster 2064 “Machine Learning — New Perspectives for Science” (ref number 390727645, PB and PH), ADIMEM (01IS18052C and 01IS18052B to PB and PH) and the Tübingen AI Center (FKZ: 01IS18039A, PB and PH). NK and PH gratefully acknowledge financial support by the German Federal Ministry of Education and Research (BMBF) through Project ADIMEM (FKZ: 01IS18052B), as well as by the European Research Council through ERC StG Action 757275 / PANAMA, and funds from the Ministry of Science, Research and Arts of the State of Baden-Württemberg.

The authors thank the International Max Planck Research School for Intelligent Systems (IMPRS-IS) for supporting Nicholas Krämer.

**Funding** Open Access funding enabled and organized by Projekt DEAL.

**Code availability** The probabilistic solvers and models were implemented in Python and Cython. The code is publicly available at <https://github.com/berenslab/neuroprobnum>.

## Declarations

**Conflict of interest** The authors have no competing interest to declare.

**Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

## References

- Abdulle, A., & Garegnani, G. (2020). Random time step probabilistic methods for uncertainty quantification in chaotic and geometric numerical integration. *Statistics and Computing*.
- Blundell, I., Plotnikov, D., Eppler, J. M., & Morrison, A. (2018). Automatically selecting a suitable integration scheme for systems of differential equations in neuron models. *Frontiers in neuroinformatics*, 12, 50.
- Bogacki, P., & Shampine, L. F. (1989). A 3 (2) pair of Runge-Kutta formulas. *Applied Mathematics Letters*, 2, 321–325.
- Borgers, C., & Nectow, A. R. (2013). Exponential time differencing for Hodgkin-Huxley-like ODEs. *SIAM Journal on Scientific Computing*, 35, B623–B643.
- Butera, R. J., & McCarthy, M. L. (2004). Analysis of real-time numerical integration methods applied to dynamic clamp experiments. *Journal of Neural Engineering*, 1, 187.
- Cash, J. R., & Karp, A. H. (1990). A variable order Runge-Kutta method for initial value problems with rapidly varying right-hand sides. *ACM Transactions on Mathematical Software (TOMS)*, 16, 201–222.
- Chen, Z., Raman, B., & Stern, A. (2020). Structure-preserving numerical integrators for Hodgkin-Huxley-type systems. *SIAM Journal on Scientific Computing*, 42, B273–B298.
- Chkrebtii, O. A., Campbell, D. A., Calderhead, B., & Girolami, M. A. (2016). Bayesian solution uncertainty quantification for differential equations. *Bayesian Analysis*, 11, 1239–1267.
- Cockayne, J., Oates, C. J., Sullivan, T. J., & Girolami, M. (2019). Bayesian probabilistic numerical methods. *SIAM Review*, 64, 756–789.
- Conrad, P. R., Girolami, M., Särkkä, S., Stuart, A., & Zygalakis, K. (2017). Statistical analysis of differential equations: introducing probability measures on numerical solutions. *Statistics and Computing*, 27, 1065–1082.



- Cranmer, K., Brehmer, J., & Louppe, G. (2020). The frontier of simulation-based inference. *Proceedings of the National Academy of Sciences*, *117*, 30055–30062.
- Dayan, P., & Abbott, L. F. (2001). Theoretical neuroscience: computational and mathematical modeling of neural systems.
- Domhof, J. W., & Tiesinga, P. H. (2021). Flexible frequency switching in adult mouse visual cortex is mediated by competition between parvalbumin and somatostatin expressing interneurons. *Neural Computation*, *33*, 926–966.
- Dormand, J. R., & Prince, P. J. (1980). A family of embedded Runge-Kutta formulae. *Journal of Computational and Applied Mathematics*, *6*, 19–26.
- Ermentrout, G. B., & Terman, D. H. (2010). The Hodgkin–Huxley equations. (pp. 1–28).
- Galán, R. F., Fourcaud-Trocmé, N., Ermentrout, G. B., & Urban, N. N. (2006). Correlation-induced synchronization of oscillations in olfactory bulb neurons. *Journal of Neuroscience*, *26*, 3646–3655.
- Gerstner, W., & Kistler, W. M. (2002). Spiking neuron models: Single neurons, populations, plasticity.
- Gerwinn, S., Bethge, M., Macke, J. H., & Seeger, M. (2008). Bayesian inference for spiking neuron models with a sparsity prior. In *Advances in Neural Information Processing Systems* (pp. 529–536).
- Gonçalves, P. J., Lueckmann, J.-M., Deistler, M., Nonnenmacher, M., Öcal, K., Bassetto, G., et al. (2020). Training deep neural density estimators to identify mechanistic models of neural dynamics. *Elife*, *9*, e56261.
- Hairer, E., Nørsett, S. P., & Wanner, G. (1993). Solving ordinary differential equations I – nonstiff problems.
- Hennig, P., Osborne, M. A., & Girolami, M. (2015). Probabilistic numerics and uncertainty in computations. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, *471*.
- Hodgkin, A. L., & Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, *117*, 500–544.
- Izhikevich, E. M. (2003). Simple model of spiking neurons. *IEEE Transactions on Neural Networks*, *14*, 1569–1572.
- Izhikevich, E. M. (2004). Which model to use for cortical spiking neurons? *IEEE Transactions on Neural Networks*, *15*, 1063–1070.
- Izhikevich, E. M. (2007). Dynamical systems in neuroscience.
- Izhikevich, E. M., & Edelman, G. M. (2008). Large-scale model of mammalian thalamocortical systems. *Proceedings of the National Academy of Sciences*, *105*, 3593–3598.
- Kersting, H., Sullivan, T. J., & Hennig, P. (2020). Convergence rates of Gaussian ODE filters. *Statistics and Computing*, *30*(6), 1791–1816.
- Krämer, N., Bosch, N., Schmidt, J., & Hennig, P. (2022, June). Probabilistic ODE solutions in millions of dimensions. In *International Conference on Machine Learning* (pp. 11634–11649). PMLR.
- Kreuz, T., Chicharro, D., Houghton, C., Andrzejak, R. G., & Mormann, F. (2013). Monitoring spike train synchrony. *Journal of neurophysiology*, *109*, 1457–1472.
- Mascagni, M. V., Sherman, A. S. et al. (1989). Numerical methods for neuronal modeling. *Methods in neuronal modeling*, *2*.
- Mulansky, M., & Kreuz, T. (2016). Pyspike—a python library for analyzing spike train synchrony. *SoftwareX*, *5*, 183–189.
- Oates, C. J., & Sullivan, T. J. (2019). A modern retrospective on probabilistic numerics. *Statistics and Computing*, *29*, 1335–1351.
- Oesterle, J., Behrens, C., Schröder, C., Hermann, T., Euler, T., Franke, K., et al. (2020). Bayesian inference for biophysical neuron models enables stimulus optimization for retinal neuroprosthetics. *Elife*, *9*, e54997.
- Oh, J., & French, D. A. (2006). Error analysis of a specialized numerical method for mathematical models from neuroscience. *Applied mathematics and computation*, *172*, 491–507.
- Papamakarios, G., Sterratt, D. C., & Murray, I. (2018). Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows. [arXiv:1805.07226](https://arxiv.org/abs/1805.07226).
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.
- Pillow, J. W., Shlens, J., Paninski, L., Sher, A., Litke, A. M., Chichilnisky, E., & Simoncelli, E. P. (2008). Spatio-temporal correlations and visual signalling in a complete neuronal population. *Nature*, *454*, 995–999.
- Prinz, A. A., Billimoria, C. P., & Marder, E. (2003). Alternative to hand-tuning conductance-based models: construction and analysis of databases of model neurons. *Journal of Neurophysiology*, *90*, 3998–4015.
- Prinz, A. A., Bucher, D., & Marder, E. (2004). Similar network activity from disparate circuit parameters. *Nature Neuroscience*, *7*, 1345–1352.
- Schober, M., Särkkä, S., & Hennig, P. (2019). A probabilistic model for the numerical solution of initial value problems. *Statistics and Computing*, *29*, 99–122.
- Stewart, R. D., & Bair, W. (2009). Spiking neural network simulation: numerical integration with the parker-sochacki method. *Journal of Computational Neuroscience*, *27*, 115–133.
- Teymur, O., Foley, C., Breen, P., Karvonen, T., & Oates, C. J. (2021). Black box probabilistic numerics. *Advances in Neural Information Processing Systems*, *34*.
- Teymur, O., Lie, H. C., Sullivan, T., & Calderhead, B. (2018). Implicit probabilistic integrators for ODEs. In *Advances in Neural Information Processing Systems* (pp. 7244–7253).
- Teymur, O., Zygalakis, K., & Calderhead, B. (2016). Probabilistic linear multistep methods. (pp. 4314–4321).
- Tronarp, F., Kersting, H., Särkkä, S., & Hennig, P. (2019). Probabilistic solutions to ordinary differential equations as nonlinear Bayesian filtering: a new perspective. *Statistics and Computing*, *29*, 1297–1315.
- Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., et al. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261–272. <https://doi.org/10.1038/s41592-019-0686-2>

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.