J Comput Neurosci
DOI 10.1007/s10827-008-0131-5

Spiking neural network simulation: numerical integration with the Parker-Sochacki method

Robert D. Stewart · Wyeth Bair

Received: 21 July 2008 / Revised: 19 November 2008 / Accepted: 25 November 2008

Abstract Mathematical neuronal models are normally expressed using differential equations. The Parker-Sochacki method is a new technique for the numerical integration of differential equations applicable to many neuronal models. Using this method, the solution order can be adapted according to the local conditions at each time step, enabling adaptive error control without changing the integration timestep. The method has been limited to polynomial equations, but we present division and power operations that expand its scope. We apply the Parker-Sochacki method to the Izhikevich 'simple' model and a Hodgkin-Huxley type neuron, comparing the results with those obtained using the Runge-Kutta and Bulirsch-Stoer methods. Benchmark simulations demonstrate an improved speed/accuracy trade-off for the method relative to these established techniques.

Keywords Parker-Sochacki · Spiking neural network · Numerical integration · Izhikevich · Hodgkin-Huxley

1 Introduction

Spiking neural network simulations are a flexible and powerful method for investigating the behaviour of neuronal systems. Spiking neuron models can be described mathematically as *hybrid systems* (Brette et al. 2007), with continuous evolution of the state variables punctuated by discrete synaptic and/or firing *events*. The continuous part of the system is generally described by a set of differential equations, and running a simulation involves repeatedly solving these equations using analytical or numerical integration methods.

The Parker-Sochacki (PS) method is a new technique for numerically integrating differential equations. PS computes iterative Talyor series expansions, enabling extraordinary integration accuracy in practical simulation time. The method is broadly applicable in computational modelling but has so far been largely overlooked in the biological sciences.

In this article, we explore the Parker-Sochacki method by applying it to two neuronal models: the Izhikevich 'simple' model (Izhikevich 2003), and a Hodgkin-Huxley neuron described in Brette et al. (2007). Benchmark simulations based on those established in Brette et al. (2007) are employed to compare the PS method with the established Runge-Kutta and Bulirsch-Stoer methods.

2 The Parker-Sochacki method

Most neuronal models can be expressed as initial value ordinary differential equations (ODEs) of the form

$$y'(t) = f(t, y); \;\; y(t_0) = y_0. \tag{1}$$

Picard's method of successive approximations was designed to prove the existence of solutions to such

Action Editor: Upinder Bhalla

R. D. Stewart (✉) · W. Bair
Department of Physiology, Anatomy and Genetics,
University of Oxford, Oxford, OX1 3PT, UK
e-mail: Robert.Stewart@pharm.ox.ac.uk

equations. The method uses an equivalent integral form for Eq. (1)

$$y(t) = y_0 + \int_{t_0}^{t} f(s, y(s))\, ds, \tag{2}$$

whose solution can be obtained as the limit of a sequence of functions $y_n(t)$ given by the following recurrence relation

$$y_{n+1}(t) = y_0 + \int_{t_0}^{t} f(s, y_n(s))\, ds. \tag{3}$$

Provided f(t,y) satisfies the Lipschitz condition locally, this sequence is guaranteed to converge locally to y. However, the iterates become increasingly hard to compute, limiting the practicality of the method in this general form.

Parker and Sochacki (1996), considered a form of Eq. (1), with $t_0 = 0$ and polynomial f. Note that the first condition is insignificant, since systems of the form of Eq. (1) can always be translated to the origin with a change of independent variable $t \rightarrow t + t_0$. Parker and Sochacki showed that polynomial f resulted in Picard iterates that were also polynomial. Furthermore, if $y_n(t)$ is truncated to degree n at each iteration, then the n-th Picard iterate is identical to the degree n Maclaurin Polynomial for $y(t)$. Using a truncated Picard iteration to compute the Maclaurin series for a polynomial ODE was termed the Modified Picard Method in Parker and Sochacki (1996), but we follow Rudmin (1998) in calling it the Parker-Sochacki method.

For a system of ODEs with all polynomial right hand sides, the PS method can be used to compute the Maclaurin series for each variable to any degree desired, thus enabling arbitrarily accurate solutions for the ODE system within the regions of convergence of the series approximations. Parker and Sochacki (1996) went on to demonstrate that a broad class of analytical ODEs can be converted into polynomial form via variable substitutions, thus rendering them solvable via the PS method. The method was subsequently extended to partial differential equations (Parker and Sochacki 2000).

Rudmin (1998) established the practical utility of the PS method by using it to solve the N-body problem in celestial mechanics. Pruett et al. (2003) developed an adaptive time-stepping version of the method for the same problem. Carothers et al. (2005) built on the algorithmic work of Rudmin to derive an efficient, algebraic PS method using Cauchy products to solve for higher order terms.

2.1 Application

To apply the PS method to a polynomial ODE system, we first define Maclaurin series for each model variable

$$y(t) = \sum_{p=0}^{\infty} y_p t^p, \tag{4}$$

with $y_0 = y(0)$, $y_1 = y'(0)$, $y_2 = \frac{y''(0)}{2!}$ and so on. Now, because the Maclaurin series is polynomial, we can write down a series for the first derivative in terms of the original series

$$y'(t) = \sum_{p=0}^{\infty} y'_p t^p = \sum_{p=0}^{\infty} (p+1) y_{(p+1)} t^p. \tag{5}$$

Equating terms, we have $y'_p = (p+1)y_{(p+1)}$. Rearranging for coefficients in the original series, we arrive at a relation that lies at the heart of the PS method

$$y_{p+1} = y'_p/(p+1). \tag{6}$$

The basis of the method is to use the model differential equations to replace y'_p with an expression in terms of the model variables. This is best illustrated through examples.

Example 1 Consider the linear system

$$\begin{aligned} y' &= y + z;\ \ y(0) = 0, \\ z' &= -y + z;\ \ z(0) = 1, \end{aligned} \tag{7}$$

Here, $y'_p = y_p + z_p$, $z'_p = -y_p + z_p$ and the PS solution is

$$\begin{aligned} y_{p+1} &= (y_p + z_p)/(p+1), \\ z_{p+1} &= (-y_p + z_p)/(p+1). \end{aligned} \tag{8}$$

Thus, each coefficient of the Maclaurin series can be computed using the previous coefficients and we can easily obtain solutions of arbitrary order. This is the general principle of the PS method.

Example 2 To demonstrate how to handle constants and higher order terms, we consider

$$y' = y^2 + 1;\ \ y(0) = 1. \tag{9}$$

The series for y and y' are as defined above, but an additional series is also defined for y^2.

$$y^2(t) = \sum_{p=0}^{\infty} \left(y^2\right)_p t^p, \tag{10}$$

with coefficients generated using Cauchy products,

$$\left(y^2\right)_p = \sum_{j=0}^{p} y_j y_{p-j}. \tag{11}$$

Since we can obtain the value of y^2 given y, we refer to y^2 as a *derived* variable, while y is a *basic* variable. The PS solution to Eq. (9) is given by

$$\begin{aligned} y_1 &= \left(y^2\right)_0 + 1, \\ y_{p+1} &= \left(y^2\right)_p / (p+1), \end{aligned} \tag{12}$$

with $p \geq 1$ and $\left(y^2\right)_p$ given by Eq. (11). Note that the constant term appears in the initial step but not in the subsequent iterations.

2.2 Simulations

In numerical simulations, the PS method is applied at each time step to solve for the system variables using initial conditions given by the solution at the previous time step. Thus, for a step size Δt, the variables are updated using truncated series approximations (up to order n), as follows:

$$y(t + \Delta t) = y(t) + \sum_{p=1}^{n} y_p (\Delta t)^p. \tag{13}$$

In a "clock-driven" simulation with fixed time step, it is always possible to rescale the system such that the effective step size becomes equal to one. Thus,

$$y(t+1) = y(t) + \sum_{p=1}^{n} y_p. \tag{14}$$

2.3 Adaptive order processing

One of the advantages of the Parker-Sochacki method is that the order of the Maclaurin series approximations depends only on the number of iterations, and can therefore be adapted according to the local conditions at each time step (Pruett et al. 2003; Carothers et al. 2005).

The PS solution is a sum over terms $y_p(\Delta t)^p$, which approximate the local truncation error for variable y on iteration p. However, with floating point numbers, rounding means that the actual change in solution at each iteration will only be approximately equal to $y_p(\Delta t)^p$. Taking this into account, we apply adaptive error control by incrementally calculating the solution and halting the iterations when the absolute changes in value of all variables are less than or equal to some error tolerance value, ε.

2.4 Variable substitutions

Equations containing exponential and trigonometric functions can often be converted into a form solvable by PS via the substitution of variables (Parker and Sochacki 1996, 2000; Carothers et al. 2005). We illustrate the method using a simple example relevant to neuronal modelling.

Example 3 Consider the system

$$\begin{aligned} y' &= z;\ \ y(0) = 1, \\ z' &= exp(y);\ \ z(0) = 1. \end{aligned} \tag{15}$$

In order to transform the system, we let $x = exp(y)$. Like y^2 in Example 3, x here is a derived variable, while y and z are basic variables. Since the derivative of an exponential function is equal to the function itself, $x' = y'x$, and the system can be rewritten:

$$\begin{aligned} y' &= z;\ \ y(0) = 1, \\ z' &= x;\ \ z(0) = 1, \\ x' &= zx;\ \ x(0) = e. \end{aligned} \tag{16}$$

2.5 Power series operations

Application of the Parker-Sochacki method can be viewed in terms of power series operations, and the examples above demonstrate all of the operations required to solve any polynomial system of ODEs. Addition and subtraction operations are applied in term-wise fashion, while the Cauchy product performs multiplication. Since integer powers can be obtained using multiplication ($y^3 = y^2 y$, $y^4 = y^2 y^2$ etc.), addition, subtraction and multiplication operations are sufficient to solve polynomial equations.

Knuth (1997) describes further power series operations that can be used to apply the Parker-Sochacki method to non-polynomial equations. First, we consider division. If we take two variables x, y, expressed as power series, and define a new variable to represent the quotient $z = x/y$, then using the Cauchy product we can write

$$x_p = \sum_{j=0}^{p} y_j z_{p-j}. \tag{17}$$

By rearrangement, we have (for $y_0 \neq 0$):

$$z_p = \left(x_p - \sum_{j=0}^{p-1} z_j y_{p-j} \right) / y_0. \tag{18}$$

Just as the Cauchy product permits variable multiplication in ODEs solvable by the PS method, this formula adds division to the list of permissible operations. Thus, the ODEs need not be strictly polynomial as suggested in prior works. Rather, PS can be applied to any equation composed only of numbers, variables, and the four basic arithmetic operations (addition, subtraction, multiplication and division), with higher powers handled through iterative multiplication.

Alternatively, Knuth (1997) presents a formula, due to Euler, for raising series to powers directly. We consider only positive integer powers here. Briefly, if $y_0 = 1$, then the coefficients of $z = y^\alpha$ are given by $z_0 = 1$ and (for $p > 0$)

$$z_p = \sum_{j=1}^{p} ((\alpha + 1) j/p - 1) y_j z_{p-j}. \tag{19}$$

For series with $y_0 \neq 1$, $z_0 = (y_0)^\alpha$ and (for $p > 0$)

$$z_p = \frac{1}{y_0} \sum_{j=1}^{p} ((\alpha + 1) j/p - 1) y_j z_{p-j}. \tag{20}$$

If the $((\alpha + 1) j/p - 1)$ terms are pre-calculated, this method uses $2p$ multiplications and a single division to calculate the p-th coefficient. Thus, Euler's power method can provide a computational saving over iterative multiplication only if more than two Cauchy products are required to calculate the power, i.e. $\alpha > 4$.

As presented, both division and general power operations run the risk of encountering division by zero. We will return to the issue of division by zero in quotient calculations in the context of the Hodgkin-Huxley neuron model in Section 4. For Euler's power method, we can circumvent the issue. If y_m is the first non-zero coefficient in y, we define a new series x, with $x_p = y_{p+m}$. Next, we take $w = x^\alpha$ and calculate the coefficients using Eq. (20). The series z has a number of leading zeros equal to the number of leading zeros in y, multiplied by the power $(m\alpha)$. Finally, $z_{p+m\alpha} = w_p$.

The presented methods for performing power series division and power operations are not new (though we are not aware of a prior description of the technique for handling leading zeros in the power calculations). However, their incorporation into the Parker-Sochacki method is both novel and powerful, significantly expanding the method's scope.

3 The Izhikevich model

The Izhikevich model (Izhikevich 2003, 2007) is a two variable, phenomenological neuron model, featuring a quadratic membrane potential, v, and a linear recovery variable, u. The model is interesting because it has simple equations yet is capable of a rich dynamic repertoire (Izhikevich 2004, 2007). The model can act as either an integrator or a resonator and can exhibit adaptation or support bursting. Indeed, this is claimed to be the simplest model capable of spiking, bursting, and being either an integrator or a resonator (Izhikevich 2007).

Subthreshold behaviour and the upstroke of the action potential can be represented as follows:

$$\begin{aligned} Cv' &= kv(v - v_t) - u + I, \\ u' &= a(bv - u), \end{aligned} \tag{21}$$

where v is the membrane potential minus the resting potential v_{rest} ($v = 0$ at rest), v_t is the threshold potential, C is the membrane capacitance, a is the rate constant of the recovery variable, k and b are scaling constants, and I is the total (inward) input current from sources other than v and u. Assuming the threshold potential is greater than the resting potential ($v_t > 0$), then when $v > v_t$, the quadratic expression in Eq. (21) will be positive, and v will tend to escape towards infinity. This escape process models the action potential upstroke. The action potential downstroke is modelled using an instantaneous reset of the membrane potential, plus a stepping of the recovery variable:

$$\text{if } v \geq v_{max} \text{ then } v \leftarrow v_{reset},\ u \leftarrow u + u_{step}, \tag{22}$$

where v_{max} is the action potential peak, v_{reset} is the post-spike reset potential, and u_{step} is used to model post-spike adaptation effects. Spike times are taken as the times when Eq. (22) is applied.

In our benchmark network simulations, synaptic interactions were modelled using a conductance-based formalism (Vogels and Abbott 2005; Brette et al. 2007). With the addition of fast excitatory (η) and inhibitory (γ) conductance-based synaptic currents, Eq. (21) becomes

$$\begin{aligned} Cv' &= kv(v - v_t) - u - \eta(v - E_\eta) - \gamma(v - E_\gamma) + I, \\ u' &= a(bv - u), \end{aligned} \tag{23}$$

where η and γ are the total excitatory/inhibitory conductances, and E_η, E_γ are corresponding reversal potentials. The conductance values are stepped by incoming synaptic events of matching type, and decay exponentially with time

$$\begin{aligned} \eta' &= -\lambda_\eta \eta, \\ \gamma' &= -\lambda_\gamma \gamma, \end{aligned} \tag{24}$$

where the λ parameters are decay rate constants.

3.1 The Parker-Sochacki solution

In this section, we develop an efficient PS solution for the Izhikevich model system Eqs. (22), (23), (24). Most calculations in the PS method require a fixed number of floating point operations at each iteration, but Cauchy products require a number of operations that scales linearly with the number of iterations. Consequently, we seek to minimise the use of Cauchy products in designing an efficient algorithm.

A straightforward solution based on Eq. (23) would require three Cauchy products: one to compute kv^2, one for ηv, and another for γv. Noting that these products contain the common factor v, we rearrange the membrane equation such that only one Cauchy product is required

$$v' = (\chi v + E_\eta \eta + E_\gamma \gamma - u + I)/C, \tag{25}$$

where $\chi = kv - \eta - \gamma - kv_t$. Thus, $\chi_0 = kv_0 - \eta_0 - \gamma_0 - kv_t$ and (for $p > 0$) $\chi_p = kv_p - \eta_p - \gamma_p$. Then the term χv is given by a Cauchy product:

$$(\chi v)_p = \sum_{j=0}^{p} \chi_j v_{p-j}. \tag{26}$$

Using this construction, an efficient Parker-Sochacki solution for the Izhikevich model can be written down as:

$$\begin{aligned}
v_1 &= \big((\chi v)_0 + E_\eta \eta_0 + E_\gamma \gamma_0 - u_0 + I\big)/C,\\
v_{p+1} &= \big((\chi v)_p + E_\eta \eta_p + E_\gamma \gamma_p - u_p\big)/(C(p+1)),\\
u_{p+1} &= a(bv_p - u_p)/(p+1),\\
\eta_{p+1} &= -\lambda_\eta \eta/(p+1),\\
\gamma_{p+1} &= -\lambda_\gamma \gamma/(p+1).
\end{aligned} \tag{27}$$

We can pre-calculate $1/C$, $1/(p+1)$ and $1/C(p+1)$ and solve using only add, subtract and multiply floating point operations.

3.2 Calculating exact spike times

In clock-driven simulations, spike times are normally restricted to discrete time samples, offering limited spike timing accuracy despite accurate integration. Furthermore, Eq. (22) implies that discretisation of spike times will dramatically affect the subsequent accuracy of the solution. Specifically, the membrane potential increases rapidly during an action potential upstroke and, because of the voltage-dependence of the recovery variable, spike timing discretisation will tend to result in significant errors in the values of both v and u prior to the application of Eq. (22). When Eq. (22) is applied, v is reset to a fixed value regardless of its prior state, but u is stepped and thus depends on its prior value. Thus, errors in the value of u are propagated through to the post-spike state. This propagation of errors can be minimised by applying Eq. (22) at the correct times.

We now show how to calculate precise spike times for the Izhikevich model despite using large time steps in simulation. To establish our method, we note the following:

1. The Maclaurin series solution for $v(t)$ is a polynomial.
2. Via a shift in v_0, locating a voltage threshold crossing can be posed as a polynomial root-finding problem.
3. Having found a supra-threshold voltage value at a discrete time point, we know that the threshold crossing must have occurred during the preceding time step.
4. Because of the escape process used to model the action potential upstroke, the membrane voltage will be monotonically increasing close to the threshold crossing/root.

Given these conditions, it is clear that we can efficiently solve this root finding problem using the Newton-Raphson method with pre-calculated polynomial coefficients.

For original step size $\Delta t = \Delta t_1$, this root-finding process returns a value Δt_{pre}, in $(0, \Delta t_1]$, reflecting the spike time within the local time step, and we solve for u, η, γ at $t + \Delta t_{pre}$. Next, Eq. (22) is applied to model the action potential downstroke and post-spike adaptation effects. Finally, an additional time step is run using the post-spike variable values as initial conditions and a step size $\Delta t_{post} = \Delta t_1 - \Delta t_{pre}$. This returns the solution to time $t + \Delta t_1$.

Figure 1 illustrates the application of this algorithm to a single neuron under constant current injection, with fixed time step ($\Delta t_1 = 0.5$ ms). In Fig. 1(a), the cell fires four times in 100 ms, with spike-rate adaptation due to the recovery variable. Figure 1(b) zooms in on the first spike, where a peak voltage crossing occurs between the 19.5 ms and 20 ms time samples. Figure 1(c) illustrates the use of the Newton-Raphson method to find the exact spike time, and the post-spike reduced step up to 20 ms is depicted in Fig. 1(d).

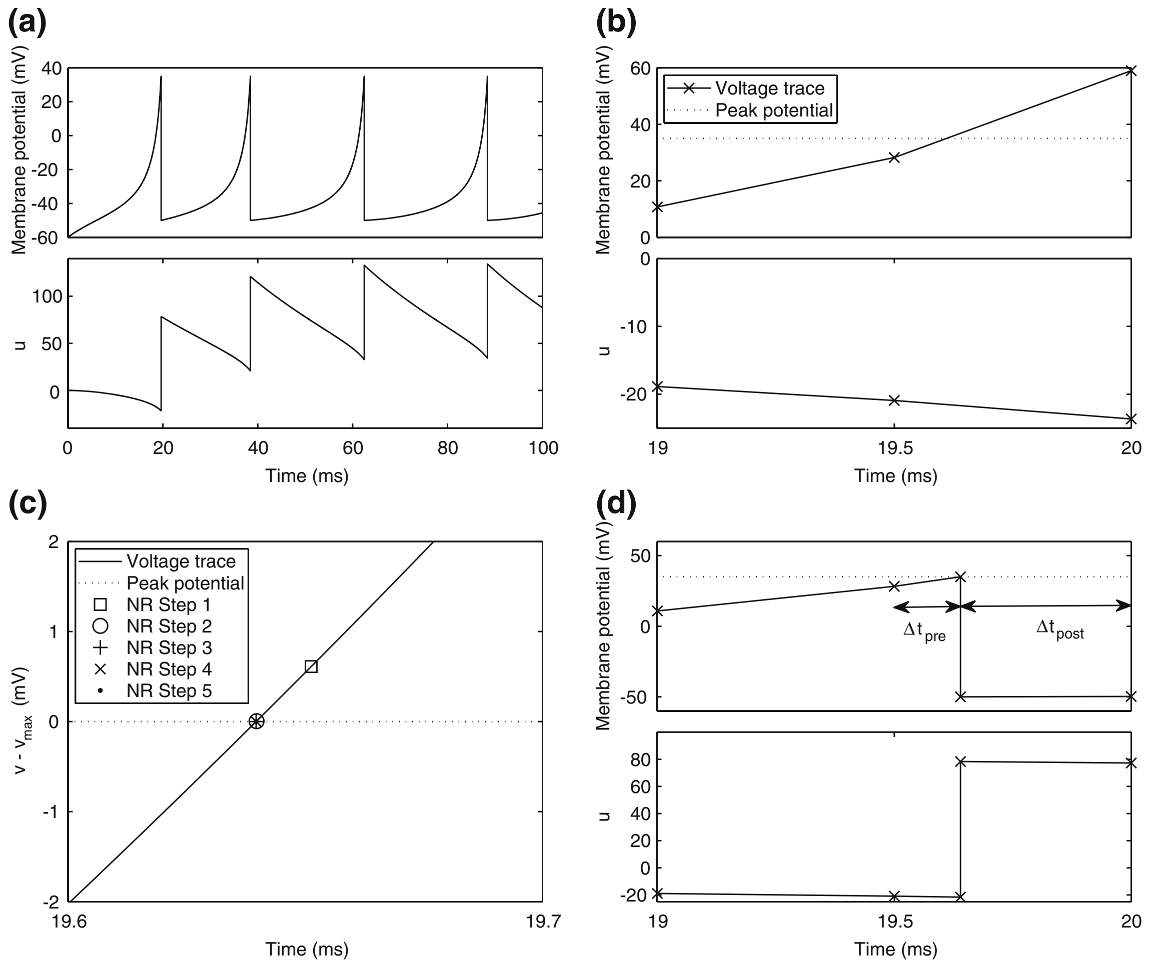

Fig. 1 Calculating exact spike times for the Izhikevich neuron. (**a**) Membrane potential (*top*) and recovery variable (*bottom*) traces over 100 ms under current injection. The true membrane potential is recovered using $v + v_{rest}$. (**b**) Twentieth millisecond of simulation using 0.5 ms standard time steps. A peak voltage threshold crossing is detected after solving for v on the second step shown. (**c**) Locating the threshold crossing using the Newton-Raphson (NR) method for root-finding. First, v_{max} is subtracted from v so that the threshold crossing becomes a root of the polynomial. Next, the Newton-Raphson method finds the root through iterative refinement. The results after steps 3–5 of this process are all contained within the circle used to plot the result from step 2, and convergence is obtained after 5 steps. (**d**) Post-spike reset and continuation. First, we solve for u at the spike time (around 19.64 ms). Next, Eq. (22) is applied to update the post-spike values of v and u. Finally, we integrate over a reduced time step ($\Delta t = \Delta t_{post}$) to arrive at the correct solution at the 20 ms time point

3.3 An adaptive order algorithm

With adaptive order processing, the complete algorithm for one Izhikevich neuron over a single time step is as follows:

1. Run Eq. (27) to order n where error checking succeeds
2. Use Eq. (13) to get a new value for v
3. If $v \geq v_{max}$
   (a) $v_0 \leftarrow v_0 - v_{max}$
   (b) Apply the Newton-Raphson method to find (Δt_{pre})
   (c) $u(t + \Delta t_{pre}) = u(t) + \sum_{p=1}^{n} u_p (\Delta t_{pre})^p$
   (d) $\eta(t + \Delta t_{pre}) = \eta(t)(e^{-\lambda_\eta \Delta t_{pre}})$
   (e) $\gamma(t + \Delta t_{pre}) = \gamma(t)(e^{-\lambda_\gamma \Delta t_{pre}})$
   (f) $v \leftarrow v_{reset}, u \leftarrow u + u_{step}$
   (g) Run a reduced time step with $\Delta t_{post} = \Delta t_1 - \Delta t_{pre}$
4. Else
   (a) update u, η, γ using Eq. (13), or Eq. (14) with rescaling

This algorithm omits synaptic events. We have developed a system for scheduling and delivering events at arbitrary, continuous time points despite using a fixed global time step, Δt_g. Provided that synaptic transmission delays are always longer than Δt_g, we can calculate in advance whether a neuron receives any synaptic events during the time interval $[t, t + \Delta t_g]$ (Morrison et al. 2007). If events are to be delivered, we move through the global step via local substeps separated by synaptic events, with each substep being processed using the algorithm presented above.

4 A Hodgkin-Huxley model

The Hodgkin and Huxley (1952) (HH) model of the squid giant axon has been arguably the most influential work in the field of computational neuroscience, and their conductance-based modelling framework remains widely employed. HH model equations are more complex than those of the Izhikevich neuron, and they are not generally expressed in a form to which the Parker-Sochacki method can be directly applied. In this section, we show how to apply the power series operations and variable substitutions described in Sections 2.5 and 2.4, to produce a PS solution algorithm.

The particular HH model neuron considered here was described in Brette et al. (2007), as a modification of a hippocampal cell model described by Traub and Miles (1991). With conductance-based synapses, the equations are,

$$\begin{aligned} Cv' &= -g_L(v - E_L) - \bar{g}_K n^4 (v - E_K) \\ &\quad - \bar{g}_{Na} m^3 h (v - E_{Na}) \\ &\quad - \eta(v - E_\eta) - \gamma(v - E_\gamma) + I, \\ n' &= \alpha_n(v)(1-n) - \beta_n(v)n, \\ m' &= \alpha_m(v)(1-m) - \beta_m(v)m, \\ h' &= \alpha_h(v)(1-h) - \beta_h(v)h, \\ \eta' &= -\lambda_\eta \eta, \\ \gamma' &= -\lambda_\gamma \gamma, \end{aligned} \tag{28}$$

where v is the membrane potential (in mV), n, m, h are gating variables for the voltage-gated sodium (m, h) and potassium (n) currents, and η and γ are excitatory and inhibitory synaptic conductances, respectively. The gating variables evolve according to voltage-dependent rate constants,

$$\alpha_n = 0.032(v_t + 15 - v)/\left[exp((v_t + 15 - v)/5) - 1\right], \tag{29}$$

$$\beta_n = 0.5\, exp((v_t + 10 - v)/40)], \tag{30}$$

$$\alpha_m = 0.32(v_t + 13 - v)/\left[exp((v_t + 13 - v)/4) - 1\right], \tag{31}$$

$$\beta_m = 0.28(v - (v_t + 40))/\left[exp((v - (v_t + 40))/5) - 1\right], \tag{32}$$

$$\alpha_h = 0.128\, exp((v_t + 17 - v)/18), \tag{33}$$

$$\beta_h = 4/\left[1 + exp((v_t + 40 - v)/5)\right], \tag{34}$$

where $v_t = -63mV$ sets the threshold (Brette et al. 2007).

4.1 The Parker-Sochacki solution

Since Eqs. (29)–(34) feature exponential functions, variable substitutions are required before PS can be applied. First, we let $a = \beta_n$ and $b = \alpha_h$. As described in Section 2.4, the new equations are as follows:

$$a' = -v'a/40, \tag{35}$$

$$b' = -v'b/18. \tag{36}$$

Equation (34) takes the form of a Boltzmann function. Now, letting $c = exp((v_t + 40 - v)/5)$, we can write

$$c' = -v'c/5. \tag{37}$$

Applying this substitution, we have $\beta_h = 4/(c+1)$, and $h' = b(1-h) - 4h/(c+1)$. Carothers et al. (2005) showed that the substitution z = 1/y yields an equation of the form $z' = -y'z^2$, and this substitution can be employed to convert β_h, and hence h', into polynomial form. However, a simpler solution is obtained via series division using Eq. (18). In this application, we let $d = h/(c+1)$, and use

$$d_p = \left(h_p - \sum_{j=0}^{p-1} d_j c_{p-j}\right)/(c_0 + 1). \tag{38}$$

Then, $h' = b(1-h) - 4d$. Note, there is no danger of encountering division by zero here since the denominator in Eq. (38) is of the form $(exp(x) + 1)$, which is always positive.

Equations (29), (31), (32) can all be written in the form x/(exp(x)−1), multiplied by some scaling constant. As with (34), we begin here by substituting for the exponential terms in the denominators of these equations. Thus, letting $e = exp((v_t + 15 - v)/5)$, $f = exp((v_t + 13 - v)/4)$, and $g = exp((v - (v_t + 40))/5)$, we have

$$e' = -v'e/5, \tag{39}$$

$$f' = -v'f/4, \tag{40}$$

$$g' = v'g/5. \tag{41}$$

Next we introduce variables for the quotient terms. Thus, $q = (v_t + 15 - v)/(e - 1)$, $r = (v_t + 13 - v)/(f - 1)$, $s = (v - (v_t + 40))/(g - 1)$, with coefficients given by:

$$q_p = \left(-v_p - \sum_{j=0}^{p-1} q_j e_{p-j} \right) / (e_0 - 1), \tag{42}$$

$$r_p = \left(-v_p - \sum_{j=0}^{p-1} r_j f_{p-j} \right) / (f_0 - 1), \tag{43}$$

$$s_p = \left(v_p - \sum_{j=0}^{p-1} s_j g_{p-j} \right) / (g_0 - 1), \tag{44}$$

yielding $\alpha_n = 0.032q$, $\alpha_m = 0.32r$, and $\beta_m = 0.28s$. There is a danger of division by zero here since the denominators in Eqs. (42)–(44) follow $(exp(x) - 1)$, which equals zero when $x = 0$. Furthermore, this condition is encountered within the normal voltage range for the model neuron. In examining the stability of the PS method around these singular points, we found that the Taylor series expansions diverged, causing the PS method to fail. To examine whether this problem was specific to the series division operation, an alternative formulation was testing using substitutions of the form $z = 1/x$, $z' = -x'z^2$ (Carothers et al. 2005). The same failures were observed here. To solve this problem, code was added to first detect series divergence and then substitute in an alternative integration method to repeat the failed step. This system was used in the HH model benchmarking simulations in Section 5.2.

As for the Izhikevich model, we simplify the membrane potential equation by grouping all the terms multiplied by v, and defining $\chi = -g_L - \bar{g}_K n^4 - \bar{g}_{Na} m^3 h - \eta - \gamma$. Thus, $\chi_0 = -g_L - \bar{g}_K (n^4)_0 - \bar{g}_{Na} (m^3 h)_0 - \eta_0 - \gamma_0$ and (for $p > 0$) $\chi_p = -\bar{g}_K (n^4)_p - \bar{g}_{Na} (m^3 h)_p - \eta_p - \gamma_p$. Similarly, $\psi = -(\alpha_n + \beta_n) = -(0.032q + a)$, $\xi = -(\alpha_m + \beta_m) = -(0.32r + 0.28s)$.

Equation (28) can now be re-written as:

$$v' = \big(\chi v + g_L E_L + \bar{g}_K n^4 E_K + \bar{g}_{Na} m^3 h E_{Na} + \eta E_\eta + \gamma E_\gamma + I\big)/C, \tag{45}$$

$$n' = \psi n + 0.032q, \tag{46}$$

$$m' = \xi m + 0.32r, \tag{47}$$

$$h' = b - bh - 4d. \tag{48}$$

The complete PS solution is listed below. Since no powers greater than four are calculated, Cauchy products are used rather than the Euler power operation. Cauchy products will comprise the major computational cost of the solution method, especially in cases where a high-order solution is required. Equation (50) uses one Cauchy product to obtain $(\chi v)_p$. Two Cauchy products are needed to obtain $(n^4)p$ (via an intermediate n^2 term). Three Cauchy products are needed for $(m^3 h)p$, via intermediate m^2 and m^3 terms. Equations (51) to (65) each require one Cauchy product to solve (in slightly modified form for the quotient variables). Thus, a total of 19 Cauchy products are required at each iteration to solve this HH model using the PS method.

$$v_1 = \big((\chi v)_0 + g_L E_L + \bar{g}_K \left(n^4\right)_0 E_K + \bar{g}_{Na} \big(m^3 h\big)_0 E_{Na} + \eta_0 E_\eta + \gamma_0 E_\gamma + I\big)/C, \tag{49}$$

$$v_{p+1} = \Big((\chi v)_p + \bar{g}_K \left(n^4\right)_p E_K + \bar{g}_{Na} \big(m^3 h\big)_p E_{Na} + \eta_p E_\eta + \gamma_p E_\gamma\Big)/(C(p+1)), \tag{50}$$

$$n_{p+1} = \big((\psi n)_p + 0.032 q_p\big)/(p+1), \tag{51}$$

$$m_{p+1} = \big((\xi m)_p + 0.32 r_p\big)/(p+1), \tag{52}$$

$$h_{p+1} = \big(b_p - (bh)_p - 4d_p\big)/(p+1), \tag{53}$$

$$\eta_{p+1} = \big(-\lambda_\eta \eta_p\big)/(p+1), \tag{54}$$

$$\gamma_{p+1} = \big(-\lambda_\gamma \gamma_p\big)/(p+1), \tag{55}$$

$$a_{p+1} = \big(-(v'a)_p/40\big)/(p+1), \tag{56}$$

$$b_{p+1} = \big(-(v'b)_p/18\big)/(p+1), \tag{57}$$

$$c_{p+1} = \big(-(v'c)_p/5\big)/(p+1), \tag{58}$$

$$d_{p+1} = \left(h_{p+1} - \sum_{j=1}^{p+1} c_j d_{p-j}\right)/(c_0+1), \tag{59}$$

$$e_{p+1} = \big(-(v'e)_p/5\big)/(p+1), \tag{60}$$

$$f_{p+1} = \big(-(v'f)_p/4\big)/(p+1), \tag{61}$$

$$g_{p+1} = \big(-(v'g)_p/5\big)/(p+1), \tag{62}$$

$$q_{p+1} = \left(-v_{p+1} - \sum_{j=1}^{p+1} e_j q_{p-j}\right)/(e_0+1), \tag{63}$$

$$r_{p+1} = \left(-v_{p+1} - \sum_{j=1}^{p+1} f_j r_{p-j}\right)/(f_0+1), \tag{64}$$

$$s_{p+1} = \left(-v_{p+1} - \sum_{j=1}^{p+1} g_j s_{p-j}\right)/(g_0+1). \tag{65}$$

With adaptive order processing, the complete algorithm for one HH neuron over a single time step is:

1. Run Eqs. (49)–(65) until error checking succeeds
2. Update variables using Eq. (13), or Eq. (14) with rescaling

For the non-power derived variables $(a-s)$, we have the option of either updating using Eq. (13) or Eq. (14), or using the definition of the variable to recalculate its value at each step. For example, for the variable c, we can use

$$c(t+\Delta t) = c(t) + \sum_{p=1}^{n} c_p (\Delta t)^p,$$

or

$$c(t+\Delta t) = exp((v_t + 40 - v(t+\Delta t))/5).$$

Using the latter method, the variable is guaranteed to match its definition at each time step. We term this method *tethering*, and any variable so updated a *tethered variable*. In preliminary testing, it was found that the stability of the PS solution was improved by tethering all the variables involved in quotient calculations (c, d, e, f, g, q, r, s), but that tethering a and b produced no improvement in the solution.

5 Results

In this section we assess the speed and accuracy of our adaptive PS algorithms by running benchmark simulations for the Izhikevich and Hodgkin-Huxley neuron models. The results are compared to those obtained using the 4th-order Runge-Kutta (RK) and Bulirsch-Stoer (BS) methods.

The 4th-order Runge-Kutta method is one of the most commonly used numerical integration methods (Press et al. 1992). The method offers moderate accuracy at moderate computational cost. For each equation, the derivative is evaluated four times per step: once at the start point, twice at trial midpoints, and once at a trial endpoint. These results are then combined in such a way that the first, second and third order error terms cancel. Thus, the solution agrees with the Taylor series expansion up to the 4th-degree term. Derivative evaluations are the major computational cost of RK.

The Bulirsch-Stoer method is another popular method. For smooth ODEs without singular points inside the integration interval, BS is described by Press et al. (1992) as the best known way to obtain high-accuracy solutions to ODEs with minimal computational effort. The method combines the (second order) modified midpoint method with the technique of Richardson extrapolation. In a single BS step, a sequence of crossings of the step is made with an increasing number n of modified midpoint substeps. Following Press et al. (1992), we use the sequence $n_k = 2k$, where k is the crossing number. After each crossing, a rational function extrapolation is carried out to approximate the solution that would be obtained if the step size were zero. The extrapolation algorithm also returns error estimates. If the latter are acceptable, we terminate the sequence and move to the next step. If not, we continue with the next crossing. For a given step size, BS can be expected to be more accurate but also more computationally expensive than RK.

Both BS and PS can apply adaptive error control without adaptive time stepping. To examine this process, adaptive stepping was not implemented for any of the methods. For PS, adaptive order processing was implemented as described in Section 2.3. Equivalent adaptive error control, based on change in the iterative solution, was employed for BS. PS was limited to a maximum of 200-th order, while BS was limited to a maximum of 50 crossings.

Simulations were run in the MATLAB 7.5 environment, with algorithms written in C and compiled as mex files. Code for the Runge-Kutta and Bulirsch-Stoer methods was adapted from the routines provided in Press et al. (1992) by removing adaptive time stepping. The machines used to run the simulations featured 2.2 GHz AMD Opteron processors, and at least 4 GB of memory. Simulation code will be provided in the ModelDB database.[1] Major routines are listed in Appendix A.

5.1 Izhikevich model

Two types of simulation were used on the Izhikevich model. In the first type, cells were driven by current injection only. In the second, recurrent synaptic interactions were also modelled. These different simulations enabled us to separate the computational costs of integration and synaptic processing.

5.1.1 Current injection simulations

The current injection model featured 1000 neurons but no functional synapses. All cells had identical parameters, set by fitting the model to the HH neuron in Brette et al. (2007). Details of the cellular model and fitting process are given in Appendix B. All simulations were of one second duration. In one series of experiments, all model cells were driven by a constant, depolarising injection current sufficient to make them fire once within the simulation time period. In another, the cells were driven to fire ten times. These are the one- and ten-spike simulations, respectively. In each case, we applied $n_c = 15$ different error tolerance conditions. For BS and PS we used a global step size of $\Delta t_g = 0.25$ ms, and systematically varied the error tolerance, ϵ. All three methods were stable (no solution divergence to infinity) at this step size. For condition c_n, $\epsilon = 1\text{e}{-}(n+1)$. For RK, we varied the error tolerance indirectly by changing the global step size. For $c_{1..15}$, Δt_g for RK was set to 1/4, 1/6, 1/8, 1/10, 1/20, 1/40, 1/60, 1/80, 1/100, 1/200, 1/400, 1/600, 1/800, 1/1000, 1/2000 ms, respectively. For time averaging, all simulations were repeated ten times. All solution algorithms included calculations for exact spike times using the Newton-Raphson method as described in Section 3.2. For RK and BS this required additional integration steps to evaluate v and v' at different time points.

Figure 2 shows the results from these simulations. In Fig. 2 (top) simulation time is plotted as a function of c. In the one-spike simulations (solid lines), PS was the fastest method for all c, with simulation times monotonically increasing from 0.72 ± 0.01 s (c_1) to 1.83 ± 0.02 s (c_{15}). RK times rose from 0.78 ± 0.01 s to 388 ± 0.7 s for $c_{1..15}$. BS times increased gradually

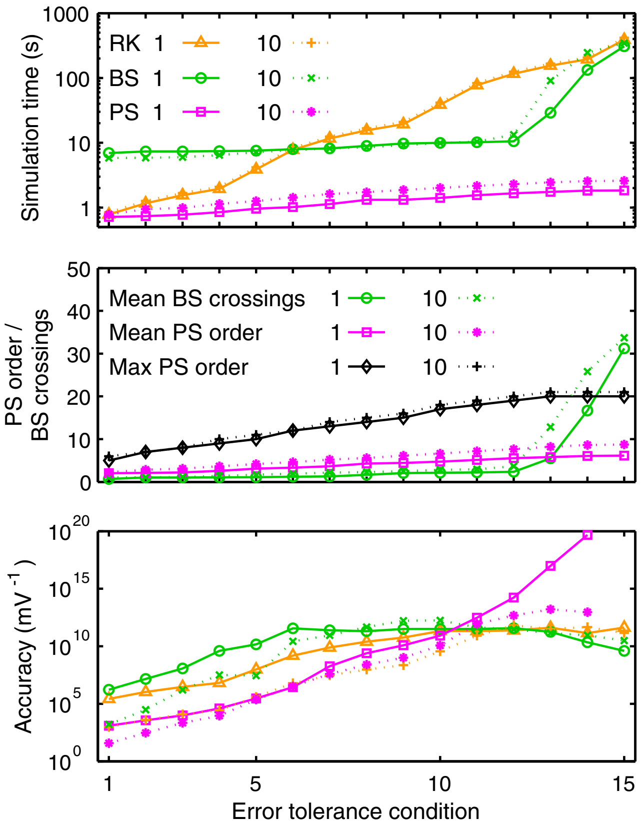

Fig. 2 Izhikevich model current injection results. (*Top*) Mean simulation time for 1 s simulations with varying error tolerance conditions. (*Middle*) Adaptive processing statistics. Plots show the mean (over a simulation) number of crossings used by the BS method per step, and the mean and maximum order of the PS method. (*Bottom*) Simulation accuracy taken as the reciprocal of absolute voltage divergence between test and reference traces. Line styles as in *top panel*

from 7 to 11 s across $c_{1..12}$ but rose steeply at tighter error tolerances to 309 ± 1 s in condition 15. Results from the ten-spike simulations were similar. PS was again the fastest method in all conditions. However, times here were greater than the one-spike results in equivalent conditions, with gain ranging from 1.09 (c_1) to 1.44 (c_7). RK times were slightly greater than for the equivalent one-spike simulations for all c, with gain factors ranging from 1.072 (c_6) to 1.085 (c_2). BS times were reduced relative to the one-spike results in the first few conditions, but were greater in the last four conditions.

In order to explain the variation in simulation times, Fig. 2 (middle) shows how adaptive processing for BS and PS varied with error tolerance by plotting representative statistics for each method. The number of BS crossings was low for $c_{1..12}$ but rose steeply for $c > 12$. This increase reflects error tolerance failures. In both one- and ten-spike simulations there were no failures

[1] http://senselab.med.yale.edu/senselab/modeldb.

for $c_{1..12}$, but for $c_{13,14,15}$ there were, respectively, 21, 475, 1666 failures per cell in the one-spike simulations and 176, 1088, and 1850 failures per cell in the ten-spike simulations. In contrast, PS never failed to achieve the specified error tolerances. The mean order of the PS method increased gradually with increasing c, and the maximum order across all conditions was 20 in one- and 21 in ten-spike simulations. The mean PS order was greater in the ten- than one-spike simulations and the gain was very similar to the simulation time gain, ranging from 1.10 (c_1) to 1.43 (c_7).

To quantify the accuracy of the simulation output, we created reference solutions against which to test all other solutions. Since the PS Taylor series were convergent in these simulations, reference solutions were obtained by running PS to complete numerical convergence ($\epsilon = 0$). There were no tolerance failures and reference simulation times were 1.84 ± 0.02 s for one spike and 2.59 ± 0.01 s for ten. Mean order was 6.14 and 8.73, respectively, and maximum order was 20 and 21 (as for c_{15}). Simulation error was calculated as the mean absolute membrane voltage divergence between test and reference traces. Figure 2 (bottom) plots simulation accuracy, taken as the reciprocal of the error.

Despite using the same error tolerance conditions, BS was many times more accurate than PS for $c_{1..10}$ in both one- and ten-spike simulations. In the one-spike simulations, RK was also more accurate than PS for $c_{1..10}$. RK was less accurate in the ten-spike simulations, but was still more accurate than PS for $c_{1..6}$. However, both BS and RK accuracy plots plateaued at low tolerances, with peak values always between 1e11 and 1e13. Indeed, BS showed reduced accuracy at the lowest tolerances where it exhibited failures. In contrast, PS showed progressive accuracy gains with decreasing tolerance until in condition 15 measured accuracy was infinite since the reference and test traces were identical for both one- and ten-spike simulations.

The reference PS simulations achieved double precision accuracy in the state variables and yet were faster than the fastest BS simulations. Furthermore, the reference runs were only 2.35 and 3.07 slower than the fastest RK simulations in the one- and ten-spike simulations, respectively.

5.1.2 Recurrent network model simulations

The network model here was based on Benchmark 1 from Brette et al. (2007), which was inspired by an earlier model (Vogels and Abbott 2005). The network featured 4000 neurons (80% excitatory, 20% inhibitory; parameters as above). All cells were randomly (2% probability) connected, and $n_s = 5$ different *network configurations* were created.

All simulations were of one second duration. To generate recurrent activity, random stimulation was applied for the first 50 ms, as described by Brette et al. (2007). This initial stimulation was provided here by constant current injection, and each cell was independently assigned a random current value in [0, 200] pA. For each network configuration, $n_i = 10$ different patterns of initial stimulation were applied, and each pairing of network configuration and input pattern defines a single *experiment* ($n_e = n_s \times n_i = 50$).

In the absence of numerical errors, all simulations from the same experiment should have produced identical output. Repeated experiments therefore allowed us to examine the speed/accuracy trade-off for each integration method.

Given the results from the current injection simulations, we selected three representative error tolerance conditions to apply here. Specifically, it was specified that $c_{1,2,3}$ here would be identical to $c_{1,9,15}$ from the current injection simulations. Thus, for BS and PS, $\epsilon =$ 1e-2, 1e-10, 1e-16, and for RK, $\Delta t_g =$ 1/4, 1/100, 1/2000. As in the current injection simulations, reference solutions were created using PS with $\epsilon = 0$.

5.1.3 Single experiment results

In this section, we characterise the outputs from a single experiment, using the reference solution to assess accuracy. Figure 3(a) shows membrane potential traces from a single neuron, with results from conditions 1–3 arranged in separate panels, top to bottom and integration methods represented using different colours. For comparison, the trace from the reference solution for this experiment is plotted as a black line in each panel. The reference trace was drawn last so that it would obscure the coloured traces when they were in agreement. Thus, working left to right in a single panel, the appearance of a coloured line is a visual indicator of divergence between the reference solution and the test solution from the method represented by that colour.

A quantitative measure of trace divergence (accuracy) was obtained by recording the time point at which each test trace first differed from the reference trace by more than 1 mV. These divergence points are indicated by vertical lines in Fig. 3(a). For c_1, the divergence times for all three methods were between 140 and 150 ms. For c_2, divergence times were later for all methods, at 433 ms (RK), 443 ms (PS), and 500 ms (BS). In the final condition, the BS and RK results were reversed relative to the previous condition with RK diverging at 500 ms,

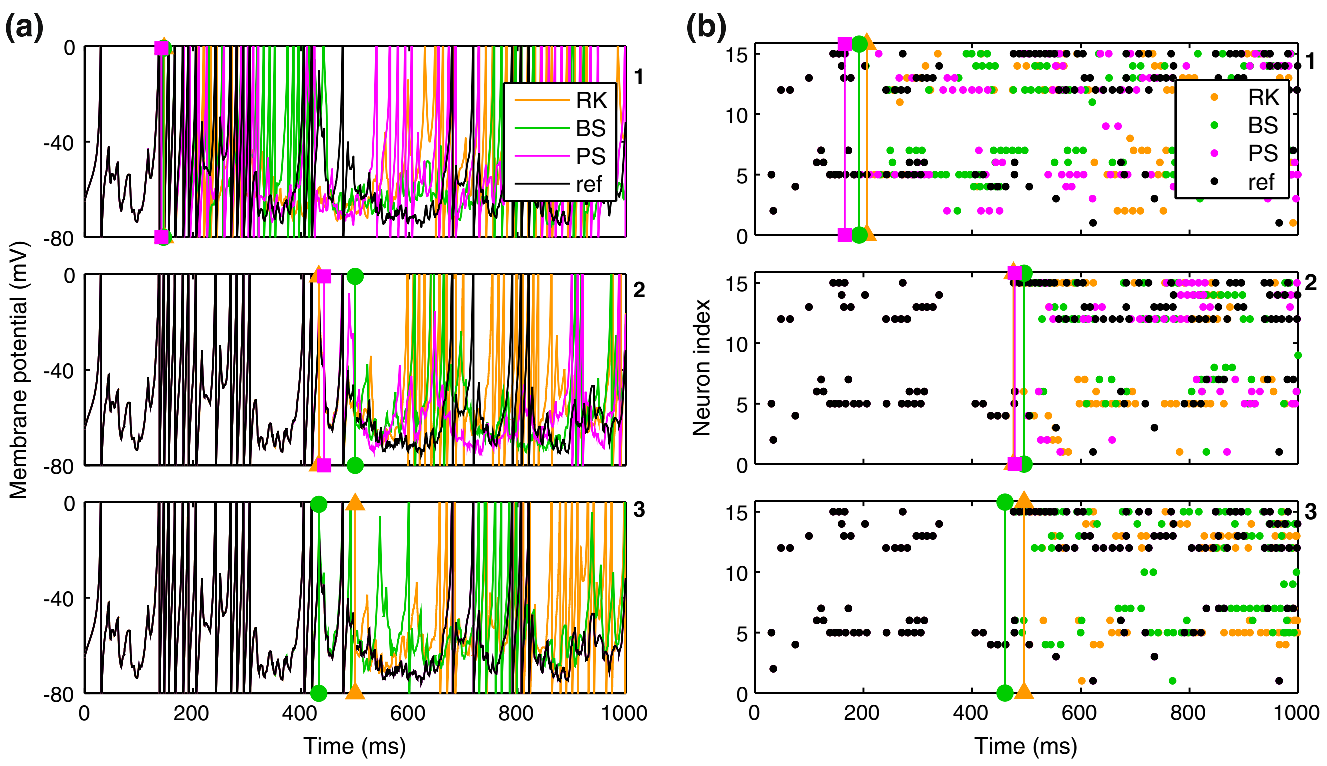

Fig. 3 Izhikevich model recurrent network results: recordings from repeated simulations using the same network model and initial random inputs but varying the integration method and error tolerance/time step size. (**a**) Membrane potential traces from single neuron. (*Top* to *bottom*) Conditions 1–3. In each plot, the colour represents the integration method, with an additional reference trace drawn in black. The reference trace was drawn last. Thus, the appearance of coloured lines indicates divergence of test solutions from the reference. (**b**) Population raster plot of the first twenty cells in the same network model as in (**a**); layout and colours as in (**a**). The single cell traced in (**a**) appears here as neuron 5. As in (**a**), these plots are overlain by a reference solution in black. Thus, the appearance of coloured dots indicates divergence

and BS diverging earlier at 433 ms. In contrast, the PS test solution agreed with the reference solution over the full one second simulation.

Figure 3(b) shows population raster plots from the first 15 cells in the same network. Once again, the reference solution is plotted in each panel to give a visual indicator of agreement. Raster divergence (vertical lines) was taken as the time point at which a test spike time first differed from the corresponding reference spike time by more than 1 ms. Raster divergence times were generally later than trace divergence times but followed the same trends. This indicates that solution divergence for this model is a network phenomenon rather than a single cell characteristic; as expected given the highly recurrent network activity here.

Both voltage traces and spike time results from each method generally converged towards the reference solution as the error tolerance decreased. The only exception to this rule was the BS result from condition 3, which was worse than condition 2. As above, BS exhibited error tolerance failures at the lowest tolerance, and reduced accuracy probably results from roundoff errors with many crossings (Press et al. 1992). Only the PS solution from condition 3 showed agreement with the reference solution that extended beyond the time limit of the simulation. RK and BS showed neither complete agreement with the reference solution, nor agreement between their own test solutions.

5.1.4 Overall performance results

In this section, we examine the overall accuracy of each integration method and derive a performance measure based on both speed and accuracy. In the single experiment results, simple heuristic measures of trace and raster plot divergence were employed to assess accuracy. While those measures were sufficient to highlight important trends in the data, a stricter measure of global solution divergence was obtained here by comparing time-ordered sequences of spikes from different simulations in the same experiment. Spike sequences consisted of {spike time, neuron index} pairs, and the duration of global solution agreement was taken as the time of the last spike at which the test and reference

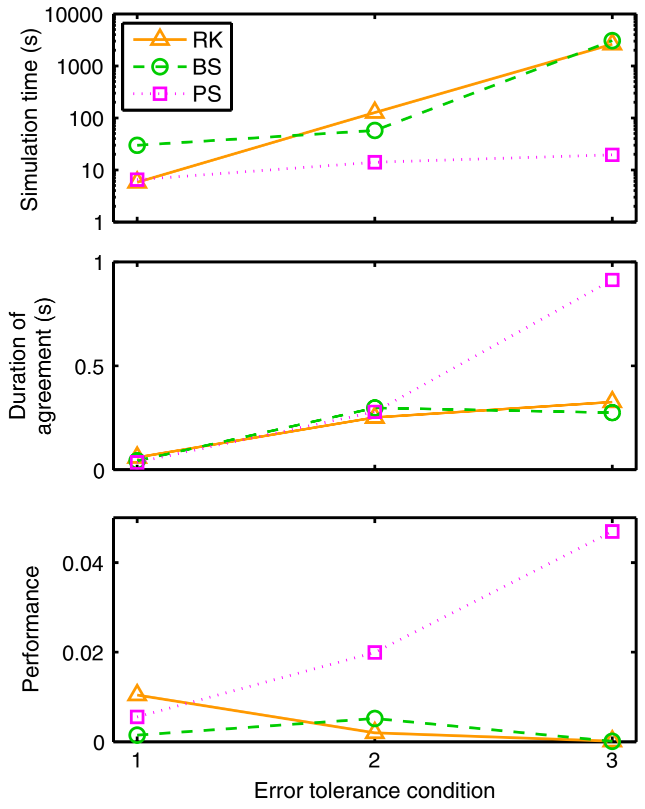

Fig. 4 Izhikevich model recurrent network results: performance summary. (*Top*) Mean simulation time. (*Middle*) Simulation accuracy: mean duration over which the output from simulation runs agree with reference solutions in terms of the exact spike sequences (see text for further details). (*Bottom*) Performance (accuracy/time)

sequence neuron indices were identical. Assuming the reference simulations were at least as accurate as the test simulations, the duration of agreement provides a measure of solution accuracy.

Figure 4 (top) plots simulation time for each method as a function of the error tolerance condition. The general pattern of results here was qualitatively similar to the current injection results in Fig. 2(a). The mean number of spikes across all experiments was 7.66 ± 0.4, so we compare with the ten-spike current injection results. RK times were between 6.07 (c_2) and 6.82 (c_1) times larger than in the equivalent current injection simulation. PS times were between 7.51 ($c_{2,3}$) and 8.31 (c_1) times larger, while BS times were between 5.09 (c_1) and 8.9 (c_3) times larger. These increased times, despite reduced firing rate, reflect the introduction of synaptic interactions, and the fact that the recurrent network had four times as many cells. Furthermore, trace recordings were output to file during the recurrent simulations but not during the current injection simulations. To examine the cost of synaptic interactions more directly, additional simulations were run with smaller recurrent networks of 1000 cells, without file recordings, and with firing rate adjusted to approximately ten spikes per cell via weight scaling. In these smaller simulations, RK times were between 0.97 ($c_{2,3}$) and 1.03 (c_1) times larger than in the equivalent current injection simulation. PS times were between 1.24 (c_1) and 1.32 (c_3) times larger, while BS times were between 1.27 (c_1) and 1.96 (c_3) times larger. Thus, the additional costs of synaptic interactions was noticeable for BS and PS, but not RK.

Figure 4 (middle) plots the mean duration of agreement for each method. These results are broadly similar to those obtained in the single experiment results. RK accuracy increased by a large amount moving from c_1 to c_2, and by a smaller amount from c_2 to c_3, reaching a maximal value of 0.33 s. BS accuracy peaked at 0.30 s for c_2 and decreased at c_3. PS accuracy increased progressively as the error tolerance was tightened. In condition 3, most, but not all, PS simulations agreed with the reference solution over the full duration of the simulation. Thus, unlike in the current injection simulations, tiny numerical differences between the simulations with $\epsilon =$ 1e-16 and $\epsilon = 0$ were sufficient here to cause network state divergence in some cases within a one second simulation time period.

Morrison et al. (2007) have argued that in order to arrive at a relevant measure of the performance of an integration method, simulation time should be analysed as a function of the integration error. Following this principle, overall performance was assessed as a function of both speed and accuracy by dividing the duration of agreement by simulation time to yield a dimensionless performance measure. For example, a performance score of 1 would be obtained by a method running at real-time speed and showing complete agreement with the reference solution. Figure 4 (bottom) plots mean performance. By this measure, RK was the best performing method for c_1, while PS performed second best for c_1 and much better than the other methods for $c_{2,3}$.

The reference PS simulations once again achieved double precision accuracy in the state variables over the simulation period and took 19.50 ± 0.7 s to run. This was faster than the fastest BS simulations, and 3.39 times slower than the fastest RK simulations.

5.2 Hodgkin-Huxley model

For the Hodgin-Huxley model, current injection simulations were used to compare methods in the absence of synaptic processing. Ten identical cells were modelled for more accurate time calculations; parameters are listed in Appendix B. As for the Izhikevich model, one- and ten-spike simulations were run and 15 error tolerance conditions were applied. Preliminary testing showed solution divergence to infinity with step sizes larger than 0.01 ms for RK, and 0.1 ms for BS/PS. Consequently, Δt_g was set to 0.1 ms for all BS and PS simulations, and 0.01 for RK in condition 1. Error tolerances for BS and PS were identical to those used in Section 5.1.1. For RK, Δt_g was reduced in the same manner as for the Izhikevich model, but from a lower starting point. Thus, for c_{15}, Δt_g for RK was 1/80,000 ms, or 12.5 ns.

For the PS method, an adaptive algorithm with failure detection was used as described in Section 4.1, with a replacement BS step being run when PS failure was detected.

Unlike the Izhikevich model simulations, the PS method sometimes failed to achieve the specified error tolerances here, giving no guarantee of accuracy. For this reason, we conducted an analysis of within-method agreement across conditions using the c_{15} results from each method as reference solutions. Voltage traces were recorded from every simulation at 1 ms sampling steps, and the mean absolute difference between test and reference traces was recorded. All methods showed initial convergence towards their own reference as the error tolerance was reduced, suggesting that they were becoming more accurate. The best result in both one- and ten-spike simulations came from PS (c_{14}) at 2.39e-13 and 1.12e-12 mV, respectively. RK and BS achieved divergences never less than 1e-10 and 1e-9 mV, respectively.

Consequently, general reference solutions for each experiment were produced using PS with $\epsilon = 0$. Treating all other voltage traces as test solutions, the mean absolute voltage difference between reference and test traces was calculated, and the inverse of this value was taken as an accuracy measure for the test solution. Finally, overall performance was taken as accuracy divided by simulation time.

Figure 5 shows the results of this performance analysis. The top panel shows how simulation time varied with the error tolerance condition. All methods became slower with decreasing error tolerance, as expected. RK was the slowest method in all conditions here. PS was the fastest method, with better than real-time speed in all conditions in the one-spike experiments. In the ten-spike simulations, PS was between 1.26 (c_1) and 3.27 (c_{15}) times slower than the equivalent one-spike simulations, but was still faster than RK and BS in all conditions.

The middle panel of Fig. 5 shows mean accuracy values plotted against the error tolerance condition. As with the Izhikevich model results, RK and BS showed

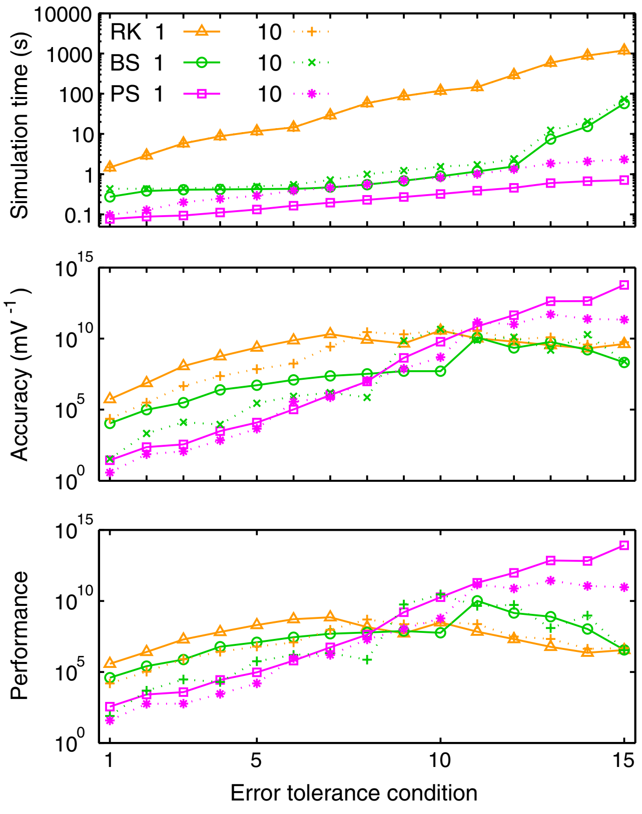

Fig. 5 HH model results. (*Top*) Mean simulation time for 1 s benchmark simulation. (*Middle*) Mean accuracy, with accuracy values taken as the inverse of the mean absolute voltage difference between test and reference solutions. (*Bottom*) Overall performance (accuracy/time)

convergence towards the reference solution with reducing error tolerance, justifying the choice of reference. As for the Izhikevich model however, the accuracy measures for both methods plateaued, reaching similar peak levels in each case, with the peak located at c_{11} (BS, one-spike) or c_{10} (all other simulations). PS accuracy continued to improve beyond this tolerance level, reaching peak values more than two orders of magnitude greater than the alternatives in the one-spike simulations and roughly one order of magnitude greater in the ten-spike experiments.

The bottom panel of Fig. 5 shows overall performance values for each method. PS recorded the best results here in both the one- and ten-spike simulations. Comparing peak performance values for each method, in the one-spike simulations PS performed roughly four orders of magnitude better than BS and five orders of magnitude better than RK. In the ten-spike simulations, PS performed 9 times better than BS, and 558 times better than RK.

The reference PS simulations took around ten percent longer to run than the c_{15} simulations in both one- and ten-spike simulations. Unlike the Izhikevich model results, double precision accuracy was not obtained due to error tolerance failures. The PS method only failed around singular points in the equations, but the replacement BS steps were unable to achieve zero error tolerance when PS failed.

6 Discussion

The Parker-Sochacki method is a promising new numerical integration technique. We have presented the method and shown how it can be applied to the Izhikevich and Hodgkin-Huxley neuronal models.

In Section 2, we summarised major milestones in the development of the Parker-Sochacki method and illustrated its application through examples. We demonstrated how to implement adaptive error control using adaptive order processing in PS. We also showed how power series division and power operations can be used within the Parker-Sochacki framework. For terms with powers greater than 4, Euler's power method can provide significant computational savings over iterated Cauchy products, but it is the division operation which is likely to be of greater utility. Series division is simple to implement since the major calculation can be carried out using a standard Cauchy product function. With this operation, PS can be directly applied to any equation composed only of numbers, variables, positive integer powers, and the four basic arithmetic operations (addition, subtraction, multiplication and division). This is a far broader class of equations than the polynomials considered in previous articles (Parker and Sochacki 1996; Carothers et al. 2005). Where other expressions are present, it may still be possible to apply the method, but additional work will be required to discover and apply suitable variable substitutions.

In Section 3, we applied PS to the Izhikevich neuron model: a simple model capable of rich dynamic behaviour. We developed an efficient PS solution using a single Cauchy product, showed how to calculate exact spike times within larger time steps using the Newton-Raphson method and presented a simple adaptive order algorithm.

Benchmark simulations in Section 5.1 demonstrated that the Parker-Sochaki method is capable of double precision integration accuracy for Izhikevich model neurons in both current injection and recurrent network simulations. Neither the Bulirsch-Stoer nor the Runge-Kutta methods were capable of the same level of accuracy. Furthermore, in Section 5.1.2 it was shown that integration accuracy had a major effect on network behaviour in a recurrent network simulation, with small solution errors leading to divergent behaviour within a relatively short simulation time period.

In light of the typical parameter uncertainties in neuronal modelling, the question of whether double precision integration accuracy is useful in a given setting will be a matter for the individual investigator to consider. However, the relative time cost of applying zero error tolerance in PS simulations is small; reference PS simulations were always faster than any BS simulations on this model and took less than four times as long to run as 4th-order RK simulations with the same global step size. In general, we would make the following suggestions. First, PS should always be run with zero error tolerance. Second, even if PS is not used for the main simulations in a study, it may still be useful as a reference solution in pilot work.

In Section 4, we applied PS to a Hodgkin-Huxley model. We showed how variable substitutions transform the equations into a form suitable for the application of PS. We also successfully applied the series division operation. However, for equations of the form $x/(exp(x)-1)$, we encountered a Taylor series divergence problem that we were unable to solve through variable substitutions. There are at least three ways to work around such a problem. First, as we did, an alternative numerical integration method can be used for steps where the PS method fails. Second, polynomial, spline, or rational function approximation methods can be used. Cubic interpolating splines are one attractive option here due to the low order and high accuracy offered (de Boor 2001). Alternatively, Floater and Hormann (2007) proposed a family of rational interpolants that have no poles, and arbitrarily high approximation orders. Finally, there exist alternative equation forms for Hodgkin-Huxley type models that avoid the presence of singular points; one promising option being the Extended Hodgkin-Huxley (EHH) model (Borg-Graham 1999) (Section 8.4.2). Here, the voltage-dependent rate constant equations take on the following generic form:

$$\alpha_x = \alpha'_x/(\tau_0(\alpha'_x + \beta'_x) + 1) \tag{66}$$

$$\beta_x = \beta'_x/(\tau_0(\alpha'_x + \beta'_x) + 1) \tag{67}$$

$$\alpha'_x = K\, exp((z\gamma(v - v_{1/2})F)/RT) \tag{68}$$

$$\beta'_x = K\, exp((-z(1-\gamma)(v - v_{1/2})F)/RT) \tag{69}$$

where K is a positive constant, F is Faraday's constant, R is the gas constant, and T is the temperature in Kelvin. The following method creates a PS solution. First, substitute for the exponential expressions to give $\alpha'_x = a_x, \alpha'_x = b_x$. Next, define a derived variable $c_x = \tau_0(a_x + b_x) + 1$. Finally, solve for α_x and β_x using series division operations on a_x/c_x and b_x/c_x, respectively.

Furthermore, since K is positive, c_x is always positive, avoiding any singularities.

In order to retain the Hodgkin-Huxley model equations used by Brette et al. (2007), benchmark simulations in Section 5.2 used the method-substitution approach. At low error tolerances, this approach appeared to yield greater accuracy than the alternative methods, but was unable to achieve double precision accuracy due to failures in both PS steps close to singular points and the replacement BS steps. Given the lack of singular points in the Extended Hodgkin-Huxley equations, we conjecture that the Maclaurin series would always be convergent for this modelling framework. If simulation testing proves this conjecture to be correct, then the PS method will be able to achieve arbitrary precision and should also run faster than on the standard Hodgkin Huxley model due to an absence of failure and replacement steps.

In all of our simulations, continuous event times were accommodated within a globally clock-driven framework. This modelling approach enables far greater accuracy than traditional clock-driven methods where events are restricted to discrete time points. Event-driven simulation approaches (Mattia and Del Giudice 2000; Delorme and Thorpe 2003; Makino 2003; Brette 2006, 2007), offer comparable event timing precision but are generally restricted to simple neuronal models, while the present approach is far more widely applicable. Morrison et al. (2007) proposed a similar hybrid clock-driven/event-driven approach but, like many standard event-driven techniques, their method was restricted to linear neuron models. In contrast, by using the Parker-Sochacki method, we were able to combine the flexibility of clock-driven simulation methods with the precision of event-driven approaches.

In Appendix A, we present the major routines used in our implementation of the Parker-Sochacki method. We endeavoured to make the code generic, modular and simple to adapt. It should be emphasized that the provided code does not constitute a general neuronal modelling package of the type described by Brette et al. (2007). Rather, it is our hope that the Parker-Sochacki method will be adopted into existing simulation packages as an alternative integration method for highly accurate simulations.

As previously noted, the Parker-Sochacki method is directly applicable to any model with polynomial or rational differential equations. In terms of existing spiking neuron models, the class with polynomial equations includes the leaky (Lapicque 1907; Tuckwell 1988) and quadratic (Ermentrout 1996; Latham et al. 2000) integrate-and-fire models, the resonate-and-fire neuron model (Izhikevich 2001), the Fitzhugh-Nagumo model (Fitzhugh 1961; Nagumo et al. 1962), and the models of spiking and bursting by Hindmarsh and Rose (1982, 1984). The exponential integrate-and-fire model (Fourcaud-Trocmé et al. 2003), including the two-dimensional adaptive version (Brette and Gerstner 2005), can be handled using an exponential variable substitution (see Section 2.4). For Hodgkin-Huxley type models, multiple substitutions will usually be required.

The Hodgkin-Huxley formalism can be viewed as a simple Markov kinetic model (Destexhe et al. 1994). More complex kinetic models have been used to model voltage-gated ion channels (see, for example Vandenberg and Bezanilla 1991; Bezanilla et al. 1994), and it has been suggested that ligand-gated, and second-messenger-gated channels can be modeled along the same lines (Destexhe et al. 1994; Destexhe 2000). Like the HH and EHH models, these general kinetic models usually feature exponential functions that can be handled using the same kind of substitutions.

Compartmental models often use equations similar to single-compartment models for each compartment, plus linear, resistive coupling terms between neighbours. The PS method handles coupling terms in the same way as any other variables; no additional substitutions or manipulations are required. Indeed, PS has already been successfully applied to the n-body problem (Rudmin 1998; Pruett et al. 2003), which features n coupling terms in the equations describing the motion of each body. Thus, the PS method is applicable to a compartmental model provided it is applicable to the equations of individual compartments, with coupling terms.

Calcium modelling introduces calcium ion concentrations as model variables, with equations describing concentration changes (Borg-Graham 1999). Once again, PS handles these new variables in the same way as any others.

In conclusion, the Parker-Sochacki method offers unprecedented integration accuracy in neuronal model simulations, at moderate computational cost, and is applicable in a variety of computational neuroscience settings. It is our hope that this article will help to facilitate its wider adoption.

Acknowledgements Research supported by The Wellcome Trust. We are grateful to Joseph Rudmin, Stephen Lucas and Jim Sochacki for stimulating discussions and helpful advice.

Appendix A: Parker-Sochacki solution code

The routines below form the core of our implementation of the Parker-Sochacki method. The generic solver routine, `ps_step`, solves a system of differential equations using PS and advances the solution across a single time step. Worker routines `first` and `iter` are passed in to `ps_step` and calculate the first and subsequent terms of the PS solution to a specific system. The `ps_step` routine was used with different `first` and `iter` functions for both Izhikevich and Hodgkin-Huxley model simulations; the Izhikevich model routines `iz_first` and `iz_iter` are included as an example. In order to use `ps_step` to solve a different model, all that is required is to define suitable ‘first’ and ‘iter’ functions specific to the model in question. Also included below are some generic power series operation functions.

```
/*Generic Parker-Sochacki solver function*/
int ps_step(double *y[],double *co[],double y1[],
  double ynew[],
  double fp[],  double eta[],
  void (*first)(double *[],double *[],double []),
  void (*iter)(double *[],double *[],double [],int),
  int stop,int ps_limit, int nv, int err_nv){
  int i,p; double dt=fp[99], dt_pow;
  first(y,co,fp); /*Calculate first order terms*/
  if(dt == 1)for(i=0;i<nv;i++)y1[i]=y[i][0]+y[i][1];
  else{for(i=0;i<nv;i++)y1[i]=y[i][0]+dt*y[i][1];
  dt_pow=dt*dt;}
  for(p=1;p<(ps_limit-1);p++){/*Iterations*/
    iter(y,co,fp,p);
    /*Update solution*/
    if(dt == 1)for(i=0;i<nv;i++)ynew[i]=y1[i]+y[i][p+1];
    else{for(i=0;i<nv;i++)ynew[i]=y1[i]+y[i][p+1]*dt_pow;
      dt_pow*=dt;
    }
    /*Check for solution divergence*/
    if((fabs(y[0][p+1])>10.0)){p=-1;break;}
    /*Test error tolerance on variable value change*/
    for(i=0;i<err_nv;i++){if(fabs(ynew[i]-y1[i])>eta[i])
    break;}
    if(i==err_nv)break;
    for(i=0;i<nv;i++)y1[i]=ynew[i];
  } p++;
  if(stop==1){if(p==ps_limit)mexErrMsgTxt
  ("PS solve failed.");}
  return p;
}

/*PS - first term for Izhikevich model*/
void iz_first(double *y[], double *co[], double fp[]){
  double v,u,g_ampa,g_gaba,I,k,l,E_ampa,E_gaba,E,a,b,
  co_g_ampa, co_g_gaba,chi;
  v = y[0][0]; u = y[1][0]; g_ampa = y[2][0];
  g_gaba = y[3][0];
  chi = y[4][0];
  I = fp[0]; k = fp[1]; E_ampa = fp[3]; E_gaba = fp[4];
  E = fp[5]; a = fp[6]; b = fp[7]; co_g_ampa = fp[8];
  co_g_gaba = fp[9];
  y[0][1] = E*(v*chi - u + E_ampa*g_ampa
   + E_gaba*g_gaba + I);
  y[1][1] = a*(b*v - u);
  y[2][1] = co_g_ampa*g_ampa;
  y[3][1] = co_g_gaba*g_gaba;
  y[4][1] = k*y[0][1] - y[2][1] - y[3][1];
}
```

```
/*PS - iteration function for higher order terms*/
void iz_iter(double *y[], double *co[], double fp[],
int p){
  double v,k,l,E_ampa,E_gaba,b,chi,vchi; int i;
  k = fp[1]; E_ampa = fp[3]; E_gaba = fp[4]; b = fp[7];
  vchi = y[0][0]*y[4][p] + y[4][0]*y[0][p];
  for(i = 1; i < p; i++){vchi += y[0][i]*y[4][p-i];}
  y[0][p+1] = co[0][p]*(vchi - y[1][p] + E_ampa*y[2][p]
  + E_gaba*y[3][p]);
  y[1][p+1] = co[1][p]*(b*y[0][p] - y[1][p]);
  y[2][p+1] = co[2][p]*y[2][p];
  y[3][p+1] = co[3][p]*y[3][p];
  y[4][p+1] = k*y[0][p+1] - y[2][p+1] - y[3][p+1];
}

/*Power series operation routines for the Cauchy product,
series division, and Euler's power method*/

void cauchy_prod(int p,double *a,double a0,double *b,
double b0,
  double *c){
  /*c is the pth term of the Cauchy product of a and b
  with  zeroth order terms a0, b0 allowing shifted
  products
  (e.g (a-1)*(b+1))*/
  int i;
  *c = a0*b[p] + b0*a[p];
  for(i = 1; i < p; i++){*c += a[i]*b[p-i];}
}

void series_div(int p,double a_pp,double *b,double b0,
  double *c,
  double c0){
  /*calculates pth term of c = a/b, with zeroth order
  terms
  b0, c0*/
  double cb;
  cauchy_prod(p,c,c0,b+1,b[1],&cb);
  c[p+1] = (a_pp - cb)/b0;
}

void series_pow(int p,double *a,double a0,double *b,
  double b0,
  double x){
  /*calculates pth coefficient of b = a^x*/
  int i;
  b[p] = x*a[p]*b0; /*i=p special case*/
  for(i=1;i<p;i++){
    b[p] += ((x+1)*(double)i/(double)p - 1)*a[i]*b[p-i];
  }
  b[p] /= a0;
}
```

Appendix B: Benchmark model neuron parameters

Cellular model parameters for both the Izhikevich and HH models were taken from Brette et al. (2007), and all cells had identical basic parameters. Briefly, the cell area was 20,000 μm^2, and input resistance was 100 MΩ. Given a specific capacitance of 1 $\mu F/cm^2$, whole cell capacitance was taken as $C = 200$ pF. Following the published code accompanying Brette et al. (2007), E_L was set to -65 mV; the value of -60 mV given in the text of the paper was erroneous (Destexhe, personal communication).

The HH neuron model was as described in Section 4. In addition to the basic parameters listed above, parameters specific to the HH model were: $g_L = 10$ nS,

$\bar{g}_{Na} = 20000$ nS, $\bar{g}_K = 6000$ nS, $E_{Na} = 50$ mV, $E_K = -90$ mV.

The Izhikevich model neuron parameters were obtained by fitting the model to the HH neuron in Brette et al. (2007). First, v_{rest} was taken as -65 mV to match E_L. The voltage threshold of -50 mV was shifted by v_{rest} to give $v_t = 15$ mV. Next, v_{max} and c were obtained by observing the HH neuron model under constant, supra-threshold current injection. The observed values of 48 mV and -85 mV were shifted relative to v_{rest} to give $v_{max} = 113$ mV and $c = -20$ mV. In the same simulations, the rheobase current was found to be around 19 pA. Since the HH neuron from Brette et al. (2007) lacks spike frequency adaptation, u_{step} was set to zero, and a was set to a value of 0.03 to match the value given by Izhikevich (2007) for a regular spiking cortical neuron.

Izhikevich (2007) (Ch 5), describes a method for setting b and k given the rheobase current, input resistance, and the resting and threshold potentials. Using this method, values of $k = 1.3$ and $b = -9.5$ were obtained.

Model synapses were conductance-based, and conductances were summed together to form one η and one γ value for each neuron. The conductances decayed exponentially with time constants of 5 ms for η and 10 ms for γ. When fired, excitatory synapses incremented η by 6 nS, while inhibitory synapses incremented γ by 67 nS.

References

Bezanilla, F., Perozo, E., & Stefani, E. (1994). Gating of shaker k+ channels: Ii. The components of gating currents and a model of channel activation. *Biophysical Journal, 66*(4), 1011–1021, April.

Borg-Graham, L. (1999). Interpretations of data and mechanisms for hippocampal pyramidal cell models. In *Cerebral cortex* (pp. 19–138). New York: Plenum.

Brette, R. (2006). Exact simulation of integrate-and-fire models with synaptic conductances. *Neural Computation, 18*(8), 2004–2027.

Brette, R. (2007). Exact simulation of integrate-and-fire models with exponential currents. *Neural Computation, 19*(10), 2604–2609.

Brette, R., & Gerstner, W. (2005). Adaptive exponential integrate-and-fire model as an effective description of neuronal activity. *Journal of Neurophysiology, 94*(5), 3637–3642, November.

Brette, R., Rudolph, M., Carnevale, T., Hines, M., Beeman, D., Bower, J. M., et al. (2007). Simulation of networks of spiking neurons: A review of tools and strategies. *Journal of Computational Neuroscience*, 23(3), 349–398, December.

Carothers, D. C., Parker, G. E., Sochacki, J. S., & Warne, P. G. (2005). Some properties of solutions to polynomial systems of differential equations. *Electronic Journal of Differential Equations, 2005*(40), 1–17.

de Boor, C. (2001). A practical guide to splines. In *Applied mathematical sciences, revised edition* (Vol. 27). New York: Springer.

Delorme, A., & Thorpe, S. J. (2003). SpikeNET: An event-driven simulation package for modelling large networks of spiking neurons. *Network, 14*(4), 613–27.

Destexhe, A. (2000). Kinetic models of membrane excitability and synaptic interactions. In J. M. Bower & H. Bolouri (Eds.), *Computational modeling of genetic and biochemical networks* (pp. 225–262). Cambridge: MIT.

Destexhe, A., Mainen, Z. F., & Sejnowski, T. J. (1994). Synthesis of models for excitable membranes, synaptic transmission and neuromodulation using a common kinetic formalism. *Journal of Computational Neuroscience, 1*(3), 195–230, August.

Ermentrout, B. (1996). Type i membranes, phase resetting curves, and synchrony. *Neural Computation, 8*(5), 979–1001, July.

Fitzhugh, R. (1961). Impulses and physiological states in theoretical models of nerve membrane. *Biophysical Journal, 1*, 445–166.

Floater, M. S., & Hormann, K. (2007). Barycentric rational interpolation with no poles and high rates of approximation. *Numerical Mathematics, 107*(2), 315–331.

Fourcaud-Trocmé, N., Hansel, D., van Vreeswijk, C., & Brunel, N. (2003). How spike generation mechanisms determine the neuronal response to fluctuating inputs. *Journal of Neuroscience, 23*(37), 11628–11640, December.

Hindmarsh, J. L., & Rose, R. M. (1982). A model of the nerve impulse using two first-order differential equations. *Nature, 296*(5853), 162–164, March.

Hindmarsh, J. L., & Rose, R. M. (1984). A model of neuronal bursting using three coupled first order differential equations. *Proceedings of the Royal Society of London. Series B, Biological Sciences, 221*(1222), 87–102, March.

Hodgkin, A. L., & Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *Journal of Physiology, 117*(4), 500–544, August.

Izhikevich, E. M. (2001). Resonate-and-fire neurons. *Neural networks, 14*(6–7), 883–894, July–September.

Izhikevich, E. M. (2003). Simple model of spiking neurons. *IEEE Transactions on Neural Networks, 14*(6), 1569–1572.

Izhikevich, E. M. (2004). Which model to use for cortical spiking neurons? *IEEE Transactions on Neural Networks, 15*(5), 1063–1070, September.

Izhikevich, E. M. (2007). *Dynamical systems in neuroscience: The geometry of excitability and bursting*. Cambridge: MIT.

Knuth, D. E. (1997). *The art of computer programming: Seminumerical algorithms* (Vol. 2, 3rd ed.). Boston: Addison-Wesley Longman.

Lapicque, L. (1907). Recherches quantitatives sur l'excitation électrique des nerfs traitée comme une polarisation. *Journal de Physiologie et de Pathologie Générale, 9*, 620–35.

Latham, P. E., Richmond, B. J., Nelson, P. G., & Nirenberg, S. (2000). Intrinsic dynamics in neuronal networks. I. Theory. *Journal of Neurophysiology, 83*(2), 808–827, February.

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

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

Morrison, A., Straube, S., Plesser, H. E., & Diesmann, M. (2007). Exact subthreshold integration with continuous spike times in discrete-time neural network simulations. *Neural Computation, 19*(1), 47–79, January.

Nagumo, J., Arimoto, S., & Yoshizawa, S. (1962). An active pulse transmission line simulating nerve axon. *Proceedings of the IRE, 50*, 2061–2070.

Parker, G. E., & Sochacki, J. S. (1996). Implementing the Picard iteration. *Neural, Parallel & Scientific Computations, 4*(1), 97–112.

Parker, G. E., & Sochacki, J. S. (2000). A Picard-Maclaurin theorem for initial value PDE's. *Abstract and Applied Analysis, 5*(1), 47–63.

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

Pruett, C. D., Rudmin, J. W., & Lacy, J. M. (2003). An adaptive N-body algorithm of optimal order. *Journal of Computational Physics, 187*(1), 298–317.

Rudmin, J. W. (1998). Application of the Parker-Sochacki method to celestial mechanics. Technical Report, James Madison University.

Traub, R. D., & Miles, R. (1991). *Neuronal networks of the hippocampus*. New York: Cambridge University Press.

Tuckwell, H. (1988). *Introduction to theoretical neurobiology: Linear cable theory and dendritic structure* (Vol. 1). Cambridge: Cambridge University Press.

Vandenberg, C. A., & Bezanilla, F. (1991). A sodium channel gating model based on single channel, macroscopic ionic, and gating currents in the squid giant axon. *Biophysical Journal, 60*(6), 1511–1533, December.

Vogels, T. P., & Abbott, L. F. (2005). Signal propagation and logic gating in networks of integrate-and-fire neurons. *Journal of Neuroscience, 25*(46), 10786–10795, November.