EFFICIENT DISCRETE EVENT SIMULATION OF SPIKING NEURONS IN NEURON

N.T. Carnevale[1]* and M.L. Hines[2]

Departments of [1]Psychology and [2]Computer Science

Yale University, New Haven, CT 06520

Abstract 312.10 AA−34

Abstract

Recent releases of NEURON can perform efficient discrete event simulations of networks of integrate−and−fire spiking neurons, as well as hybrid simulations involving both integrate−and−fire neurons and cells with voltage−gated conductances. This is made possible by NEURON's event delivery system, which opens up a large domain of problems in which certain types of "artificial" spiking cells, and networks of them, can be simulated hundreds of times faster than with numerical integration. Discrete event simulations are possible when all state variables of a model cell can be computed analytically from a new set of initial conditions. Computations are performed only when an event occurs, so total computation time is proportional to the number of events delivered, and is independent of the problem time and the numbers of cells and connections. Thus a simulation that involves 10^5 spikes in 1 hour for 100 cells takes the same time as one with 10^5 spikes in 1 second for 1 cell. The three classes of integrate−and−fire neurons built into NEURON are leaky integrators that differ in their response to input events. An input of weight *w* to an IntFire1 cell makes its "membrane potential" jump instantaneously by that amount. IntFire2 integrates a steady bias current plus a net synaptic current with first order kinetics that is driven by input events. Excitatory events to IntFire3 drive a "depolarizing" current with fast first order kinetics, while inhibitory events drive a "hyperpolarizing" current with slower, second order kinetics.

Introduction

The NEURON simulation environment was initially developed to handle neuronal models in which complex membrane properties and extended geometry play important roles (Hines 1989; 1993; Hines and Carnevale 1995). However, NEURON has continued to evolve to address the evolving research needs of experimental and theoretical neuroscientists. For most of the past decade it has been used to model networks of biological neurons, e.g. (Destexhe et al. 1993; Lytton et al. 1997; Sohal et al. 2000). This work stimulated the development of powerful strategies that increase the convenience and efficiency of creating, managing, and exercising such models (Destexhe et al. 1994; Lytton 1996; Hines and Carnevale 2000). Further enhancements have been prompted by increasing research on networks of spiking neurons, e.g. (Maas and Bishop 1999; Riecke et al. 1997), so that the most recent releases of NEURON are capable of efficient discrete event simulations of networks of "artificial" (integrate and fire) spiking

neurons, as well as hybrid simulations of nets whose elements include both artificial neurons and neuron models with membrane currents governed by voltage−gated ionic conductances. Here we show how discrete events are used in NEURON to implement three broad classes of integrate and fire neurons.

NEURON's event delivery system

NEURON's event delivery system opens up a large domain of discrete event simulations in which certain types of "artificial" spiking cells, and networks of them, can be simulated hundreds of times faster than with numerical integration. Discrete event simulations are possible when all state variables of a model cell can be computed analytically from a set of initial conditions. That is, if an event occurs at time t_1, all state variables must be computable from the state values and time t_0 of the previous event. Since computations are performed only when an event occurs, run time is proportional to the number of events delivered and independent of the number of cells, number of connections, or problem time. Thus handling 100,000 spikes in one hour for 100 cells takes the same time as handling 100,000 spikes in 1 second for 1 cell.

The NetCon (Network Connection) class is the portion of the event delivery system that is used to define connections between cells. A NetCon object watches its source cell for the occurrence of a spike event and then, after some time delay, delivers a weighted synaptic input event to a target cell. That is, the NetCon object represents axonal spike propagation and synaptic delay. More generally, it can be thought of as a channel that transmits a stream of events from a source to a target. The implementation of this service takes into account the fact that the delay between initiation and delivery of events is different for different streams. Consequently the order in which events are generated by a set of sources is rarely the order in which they are received by a set of targets. Furthermore the delay may be anything in the range $[0, 10^9]$.

Three classes of integrate and fire cells

Recent releases of NEURON have three broad classes of integrate and fire cells built in. Handling of incoming events and the calculations necessary to generate outgoing events are specified using the `NET_RECEIVE` block of NEURON's model description language NMODL (Hines and Carnevale 2000). Artificial cells are implemented as point processes that serve as both targets and

sources for NetCon objects. They are targets because they have a `NET_RECEIVE` block, which handles discrete event input streams through one or more NetCon objects. They are also sources because the `NET_RECEIVE` block also generates discrete output events which are delivered through one or more NetCon objects.

> Computer code listings have been edited to remove unnecessary detail for the sake of clarity; omissions are marked by ellipsis . . . and *italics*. Complete source code for all mechanisms described here are included with NEURON, which is available at no charge from `http://www.neuron.yale.edu`

IntFire1: a basic integrate and fire model

NEURON's simplest built−in integrate and fire mechanism is IntFire1, which has a "membrane potential" state *m* which decays toward 0 with time constant τ.

$$\tau \frac{dm}{dt} + m = 0 \qquad \text{Eq. 1}$$

An input event of weight *w* adds instantaneously to *m*, and when $m \geq 1$ the cell "fires," producing an output event and returning *m* to 0. Negative weights are inhibitory while positive weights are excitatory. This is analogous to an electrotonically compact cell whose membrane time constant τ is very long compared to the time course of individual synaptic conductance changes. Every synaptic input quickly shifts membrane potential to a new level, and each cell firing erases all traces of prior inputs. **Listing 1** shows an initial implementation of this model.

The response of an IntFire1 cell with τ = 10 ms to input events is shown in **Fig. 1**. The events arrive at *t* = 5, 22, and 25 ms, each with a weight *w* = 0.8. The third input triggers a spike. The plot of `m` in **Fig. 1 top** looks like a staircase because this variable is evaluated only when a new input event arrives. A function can be included in the mod file that defines IntFire1 to give a better indication of the time course of the integration state *m*. Plotting this function during a simulation with fixed time steps (`dt` = 0.025 ms, **Fig. 1 middle**) demonstrates the exponential decay of *m* between input events. In a simulation run with variable time steps (**Fig. 1 bottom**), the decay appears to follow a sequence of linear ramps. This is only an artifact of the Graph tool drawing lines between the points computed analytically at the time steps chosen by the integrator.

> Note: Many of these graphs show smooth plots to facilitate visualization of integrate and fire mechanisms, e.g. `M` in Fig. 1.

However, we must emphasize that the simulation calculations are analytic and are performed only at event arrival, regardless of esthetic graphical refinements.

Adding a relative refractory period to IntFire1 is as simple as initializing *m* to a negative value after the cell fires. Alternatively, a depolarizing afterpotential can be emulated by initializing *m* to values in the range (0, 1).

The IntFire1 built into NEURON has an absolute refractory period that makes use of a special feature of NEURON's event delivery system called self−events. A `PARAMETER` named `refrac` specifies the duration of the refractory period and an `ASSIGNED` variable called `refractory` keeps track of whether or not the mechanism is in the refractory period.

Listing 2 shows a `NET_RECEIVE` block that implements an absolute refractory period. If `refractory` equals 0, the cell accepts external events (i.e. events delivered by a NetCon) and calculates the state and whether to fire the cell. When the cell fires a spike, `refractory` is set to 1 and further external events are ignored. The `flag` variable is a keyword which is defined as 0 when an external event is received. If its value is non−zero, it must have been set by a call to `net_send()` when the cell fired. The `net_send(interval, flag)` statement places an event into the delivery system as an "echo" of the current event, i.e. it will come back to the sender after the specified `interval` and with the specified `flag`. In this case we aren't interested in the weight but only the `flag`. When this self−event comes back, it means that the refractory period is over.

Figure 2 illustrates an IntFire1 with a refractory interval of 5 ms subjected to a train of inputs with weight $w = 0.4$ at 3 ms intervals (arrows). The fourth input, which occurs at 11 ms, drives the cell above firing threshold. The `M` function of this mechanism imitates the appearance of a spike by following a suggestive stereotyped time course, but it should be recalled that this is a purely cosmetic feature that has nothing to do with the computation of the actual state *m*. During the refractory interval, the cell is unresponsive to further inputs. At 16 ms `refractory` falls to 0, as does `M`, and the cell is once again responsive to input events.

Sending an event to oneself involves very little overhead, yet it allows elaborate calculations to be performed much more efficiently than if they were executed on a per `dt` basis. This is exploited in the implementation of two other built−in integrate and fire mechanisms that offer greater kinetic complexity than IntFire1.

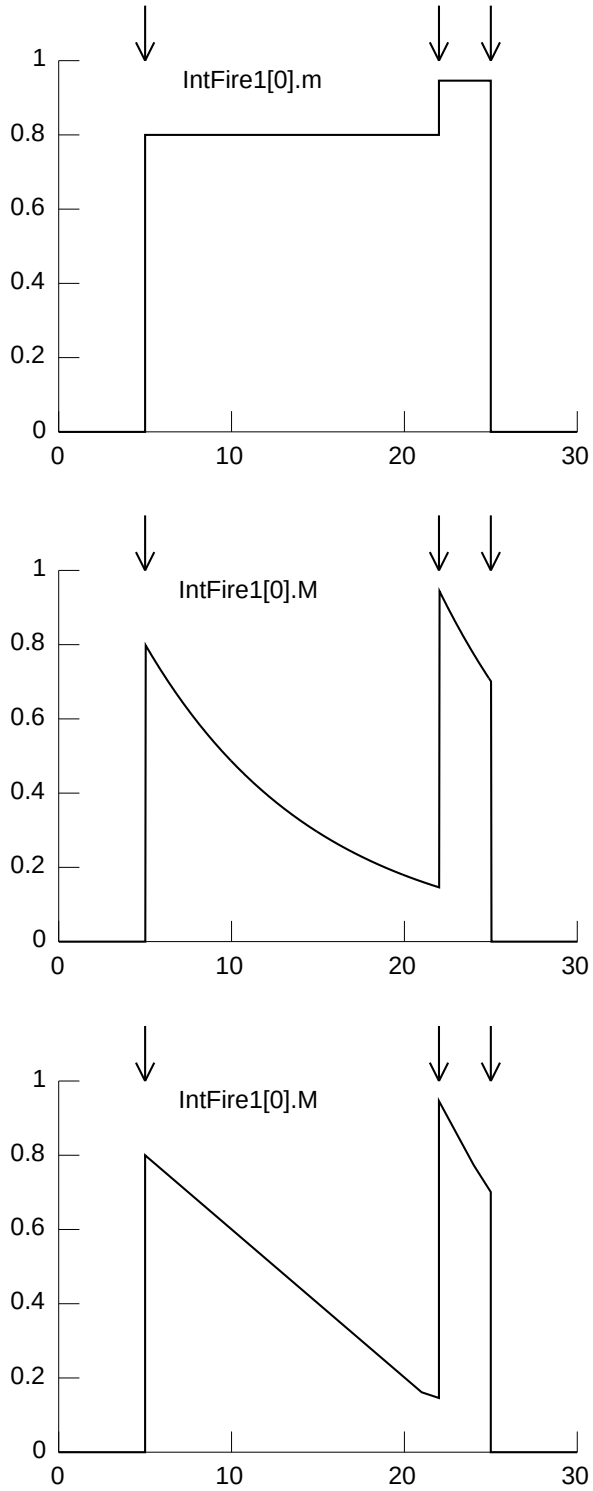
Listing 1: Initial implementation of IntFire1

```
NEURON {
    POINT_PROCESS IntFire1
    RANGE tau, m
}

PARAMETER { tau = 10 (ms) }

ASSIGNED {
    m
    t0 (ms)
}

INITIAL {
    m = 0
    t0 = 0
}

NET_RECEIVE (w) {
    : calculate present m analytically
    m = m*exp(-(t - t0)/tau)
    : increment by weight of event
    m = m + w
    : keep track of last event time
    t0 = t
    if (m >= 1) {
        : threshold exceeded
        net_event(t)
        m = 0
    }
}
```

Calculations take place here only when a NetCon delivers a new event. There is no `BREAKPOINT` or `SOLVE` block to be executed at every `dt`

Notify all NetCon objects for which this point process is a source that it fired a spike at time `t`. Then reset `m` to 0.

Figure 1



Listing 2: Adding refractoriness to IntFire1

```
NET_RECEIVE (w) {
  if (refractory == 0) {
    : accept external events
    m = m*exp(−(t − t0)/tau)
    m = m + w
    t0 = t
    if (m >= 1) {
      net_event(t)
      refractory = 1
      net_send(refrac, refractory)
      m = 2 : imitation "spike"
    }
  } else if (flag == 1) {
    refractory = 0
    m = 0
    t0 = t
  } : else ignore the external event
}
```

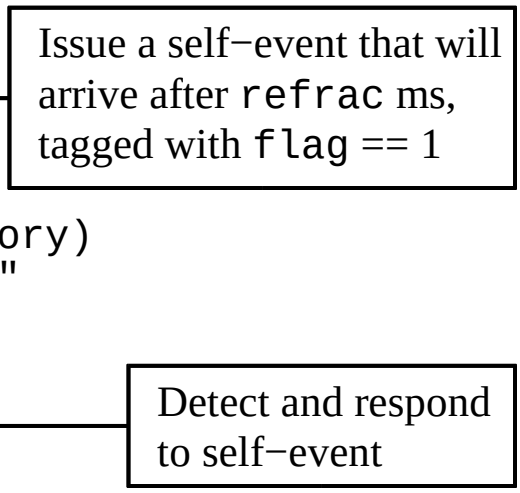
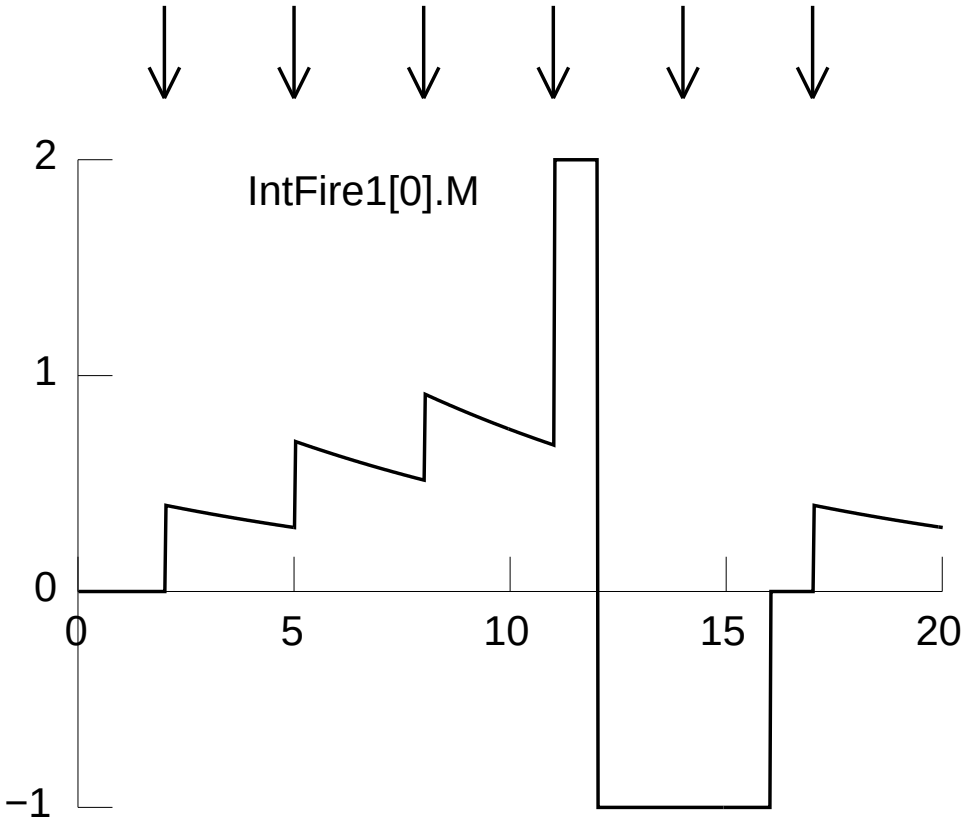


Figure 2



IntFire2: firing rate proportional to input

The IntFire2 mechanism, like IntFire1, has a "membrane potential" state *m* that follows first order kinetics with time constant τ_m. However, an input event to IntFire2 does not have an immediate effect on *m*. Instead it produces a discontinuous change in the net synaptic current *i*. Between events, *i* decays with time constant τ_s toward a steady input current of magnitude i_b. That is,

$$\tau_s \frac{di}{dt} + i = i_b \qquad \text{Eq. 2}$$

where an input event causes *i* to change abruptly by *w* (**Fig. 3 top**). This piecewise continuous current *i* drives *m* so that

$$\tau_m \frac{dm}{dt} + m = i \qquad \text{Eq. 3}$$

where $\tau_m < \tau_s$. Thus an input event produces a more gradual change in *m* that is described by two time constants and approximates an alpha function if $\tau_m \approx \tau_s$. When *m* crosses a threshold of 1 in a positive direction, the cell fires, *m* is reset to 0, and integration resumes immediately (**Fig. 3 bottom**). Note that *i* is not reset to 0, i.e. cell firing does not obliterate all traces of prior synaptic activation, as it did in the IntFire1 mechanism.

Depending on its parameters, IntFire2 can emulate a wide range of relation−ships between input pattern and firing rate. The firing rate is approximately i / τ_m if *i* is >> 1 and changes slowly compared to τ_m.

The i_b current is analogous to the combined effect of a baseline level of synaptic drive plus a bias current injected through an electrode. The requirement that $\tau_m < \tau_s$ is equivalent to asserting that the membrane time constant is faster than the decay of the current produced by an individual synaptic activation. This is plausible for slow inhibitory inputs, but where fast excitatory inputs are concerned an alternative interpretation can be applied: each input event signals an abrupt increase (followed by an exponential decline) in the mean firing rate of one or more afferents that produce brief but temporally overlapping postsynaptic currents. The resulting change of *i* is then the moving average of these currents.

The IntFire2 mechanism is amenable to discrete event simulations because Equations 2 and 3 have analytic solutions. If the last input event occurred at time t_0 and the values of i and m immediately after that event were $i(t_0)$ and $m(t_0)$, then their subsequent time course is given by

$$i(t) = i_b + [i(t_0) - i_b]\, e^{-(t-t_0)/\tau_s} \qquad \text{Eq. 4}$$

and

$$m(t) = i_b + \left[i(t_0) - i_b \right] \frac{\tau_s}{\tau_s - \tau_m} e^{-(t-t_0)/\tau_s} + \left\{ m(t_0) - i_b - \left[i(t_0) - i_b \right] \frac{\tau_s}{\tau_s - \tau_m} \right\} e^{-(t-t_0)/\tau_m} \qquad \text{Eq. 5}$$

At the core of the implementation of IntFire2 is the function `firetime()`, which is discussed below. This function projects when m will equal 1 based on the present values of i_b, i, and m, assuming that *no new input events arrive*. The value returned by `firetime()` is 10^9 if the cell will never fire with no additional input. Note that if $i_b > 1$ the cell fires spontaneously even if no input events occur.

NMODL syntax includes an `INITIAL` block (**Listing 3**) whose statements are executed when a simulation is initialized (Hines and Carnevale 2000). The `INITIAL` block in IntFire2 calls `firetime()` and uses the returned value to put a self−event into the delivery system. The strategy, which is spelled out in the `NET_RECEIVE` block, is to respond to input ("external") events by moving the delivery time of the self−event back and forth with the `net_move()` function. When the self−event is finally delivered (potentially never), `net_event()` is called to signal that this cell is firing. Notice that external events are never ignored (and shouldn't be even if we introduced a refractory period where we refused to integrate m) but always have an effect on the value of i.

The function `firetime()` returns the first $t \geq 0$ for which

$$a + b\, e^{-t/\tau_s} + (c - a - b)\, e^{-t/\tau_m} = 1 \qquad \text{Eq. 6}$$

where the parameters a, b and c are defined by the coefficients in Eq. 5a. If there is no such t the function returns 10^9. This represents the time of the next cell

firing, relative to the time t_0 of the most recent synaptic event. Since this computation must be performed on every input event, it is important to minimize the number of Newton iterations.

For this we use a strategy that depends on the behavior of the function

$$f_1(x) = a + b\,x^r + (c - a - b)\,x \text{ where } x = e^{-t/\tau_m} \text{ and } r = \tau_m/\tau_s \qquad \text{Eq. 7a}$$

over the domain $0 < x \leq 1$. Note that $c < 1$ is the value of f_1 at $x = 0$ (i.e. at $t = \infty$). This function f_1 has the advantage of being either linear in x (if $b = 0$) or convex up ($b > 0$) or down ($b < 0$) with no inflection points. Since $r < 1$, f_1 is tangent to the y axis for any nonzero b (i.e. $f_1'(0)$ is infinite).

Figure 4 top illustrates the qualitative behavior of f_1 for $a \leq 1$. It is easy to analytically compute the maximum in order to determine if there is a solution to $f_1(x) = 1$. If a solution exists, f_1 is concave downward so Newton iterations starting at $x = 1$ underestimate the firing time.

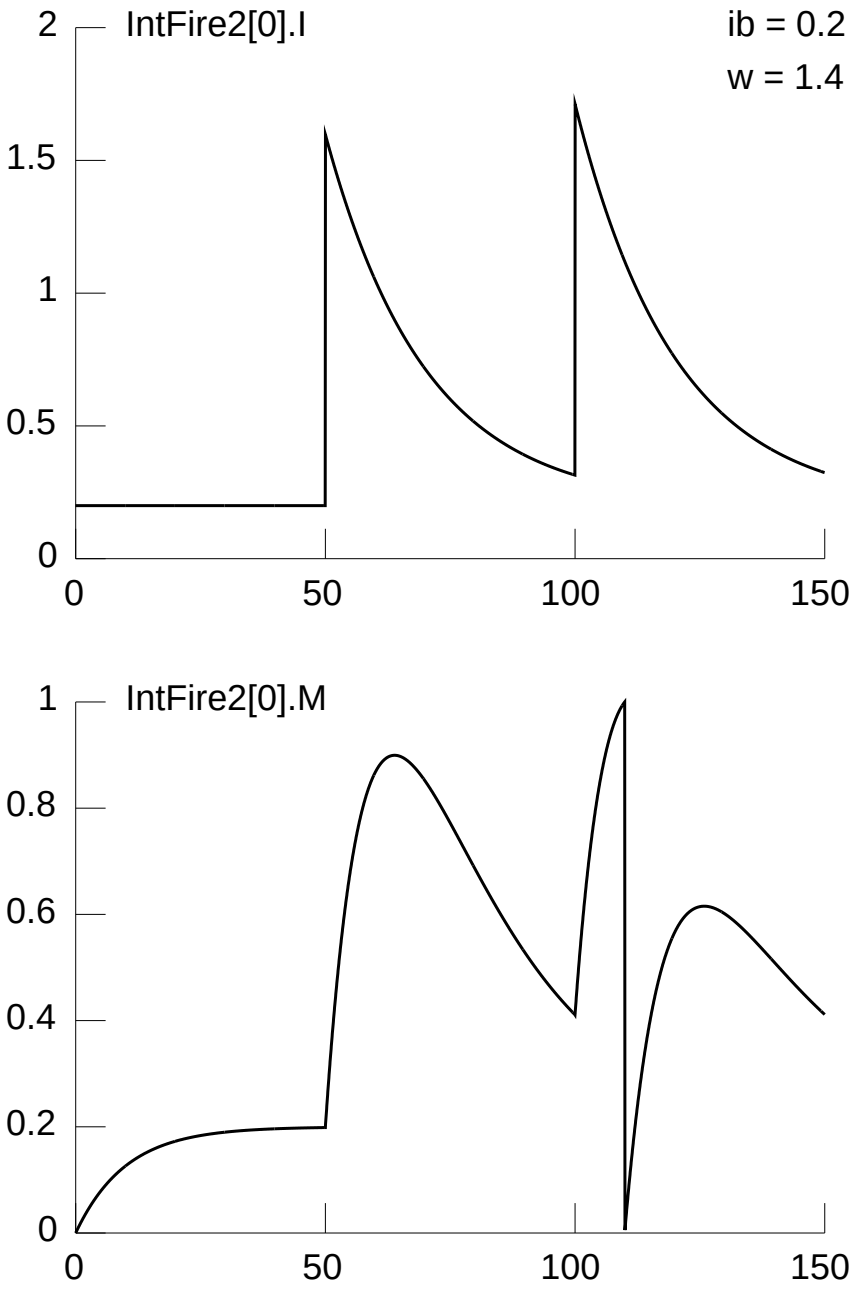
For $a > 1$, a solution is guaranteed. However, Newton iterations starting at $x = 1$ are inappropriate if the slope there is more negative than $c - 1$ (dashed line in **Fig. 4 middle**). In that case, the transformation $x = e^{-t/\tau_s}$ is used, giving the function

$$f_2(x) = a + b\,x + (c - a - b)\,x^{1/r} \qquad \text{Eq. 7b}$$

and the Newton iterations begin at $x = 0$ (**Fig. 4 bottom**).

Since iterations are performed over regions in which f_1 and f_2 are relatively linear, `firetime()` usually requires only two or three Newton iterations to converge to the next firing time. The only exception is when f_1 has a maximum that is just slightly larger than 1, in which case it may be a good idea to stop after a couple of iterations and issue a self−event. The advantage of this strategy is that it defers a costly series of iterations and allows an interval in which another external event might arrive that would force computation of a new projected firing time. Such an event, regardless of whether excitatory or inhibitory, would very likely make it easier to compute the next firing time.

Figure 3

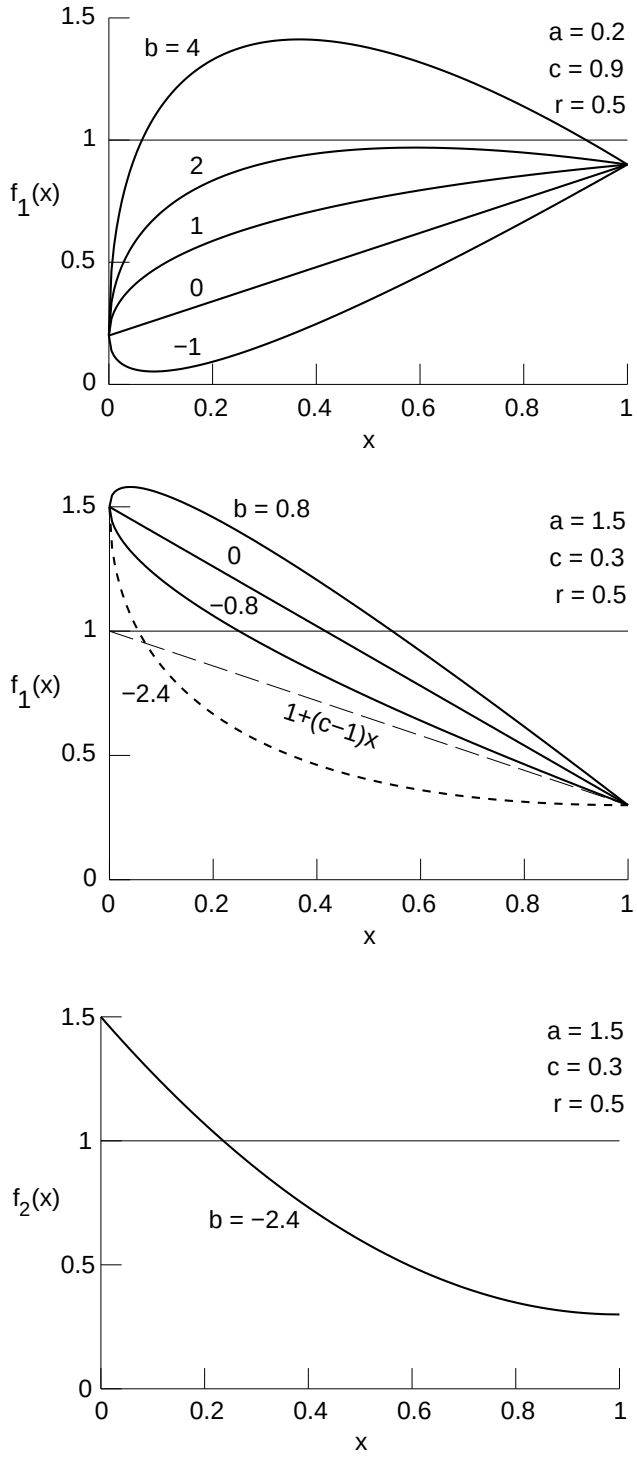


τ_m = 10 ms, τ_s = 20 ms, i_b = 0.2, w = 1.4

Listing 3: IntFire2

```
INITIAL {
  . . .
  net_send(firetime(args), 1)
}

NET_RECEIVE (w) {
  . . .
  if (flag == 1) { : time to fire
    net_event(t)
    m = 0
    . . .
    net_send(firetime(args), 1)
  } else {
    . . .
    compute new value of m
    if (m >= 1) {
      net_move(t) : fire now
    } else {
      . . .
      net_move(firetime(args) + t)
    }
  }
  update t0 and assign new value to i
}
```

Figure 4


1.5
1
0.5
0
$f_1(x)$
b = 4
2
1
0
−1
a = 0.2
c = 0.9
r = 0.5
0
0.2
0.4
0.6
0.8
1
x
b = 0.8
0
−0.8
−2.4
1+(c−1)x
a = 1.5
c = 0.3
r = 0.5
$f_2(x)$
b = −2.4

IntFire4: different synaptic time constants

While the dynamics of IntFire2 are more complex than IntFire1, they are restricted in that the response to any external event, whether excitatory or inhibitory, has the same kinetics. As we pointed out in the discussion of IntFire2, it is possible to interpret excitatory events in a way that partially sidesteps this issue. However, experimentally observed synaptic excitation tends to be faster than inhibition so a more flexible integrate and fire mechanism is needed.

The IntFire4 mechanism addresses this need. Its dynamics are specified by four time constants: τ_e for a fast excitatory current, τ_{i1} and τ_{i2} for a slower inhibitory current, and τ_m for the even slower leaky "membrane" which integrates these currents. When *m* reaches 1, the cell "fires," producing an output event and returning *m* to 0. This does not affect the other states of the model.

IntFire4 is governed by the differential equations

$$\frac{de}{dt} = -k_e\, e \qquad \text{Eq. 8}$$

$$\frac{di_1}{dt} = -k_{i1}\, i_1 \qquad \text{Eq. 9}$$

$$\frac{di_2}{dt} = -k_{i2}\, i_2 + a_{i1}\, i_1 \qquad \text{Eq. 10}$$

$$\frac{dm}{dt} = -k_m\, m + a_e\, e + a_{i2}\, i_2 \qquad \text{Eq. 11}$$

where each *k* is a rate constant that equals the reciprocal of the corresponding time constant, and it is assumed that $k_e > k_{i1} > k_{i2} > k_m$. An input event with weight $w > 0$ (i.e. an excitatory event) adds instantaneously to the excitatory current *e*. Equations 9 and 10, which define the inhibitory current i_2, are based on the reaction scheme

$$i_1 \xrightarrow{k_{i1}} i_2 \xrightarrow{k_{i2}} bath \qquad \text{Eq. 12}$$

in which an input event with weight $w < 0$ (i.e. an inhibitory event) adds

instantaneously to i_1. The constants a_e, a_{i1}, and a_{i2} are chosen to normalize the response of the states e, i_1, i_2, and m to input events (**Fig. 5**). Therefore an input with weight $w_e > 0$ (an "excitatory" input) produces a peak e of w_e and a maximum "membrane potential" m of w_e. Likewise, an input with weight $w_i < 0$ (an "inhibitory" input) produces an inhibitory current i_2 with a minimum of w_i and drives m to a minimum of w_i. Details of the analytic solution to these equations are presented in an Appendix which can be obtained from `http://www.neuron.yale.edu/neuron/bib/nrnpubs.html`

IntFire4, like IntFire2, finds the next firing time through successive approximation. However, IntFire2 generally iterates to convergence every time an input event is received, whereas the algorithm used by IntFire4 exploits the convexity of the trajectory of m so that single Newton iterations alternating with self−events converge to the correct firing time. Specifically, if an event arrives at time t_0, then values of $e(t_0)$, $i_1(t_0)$, $i_2(t_0)$, and $m(t_0)$ are calculated analytically. Should $m(t_0)$ be subthreshold, the self−event is moved to a new approximate firing time t_f that is based on the slope approximation to m

$$t_f = t_0 + (1 - m(t_0)) / m'(t_0) \quad \text{if} \quad m'(t_0) > 0 \qquad \text{Eq. 13}$$

or

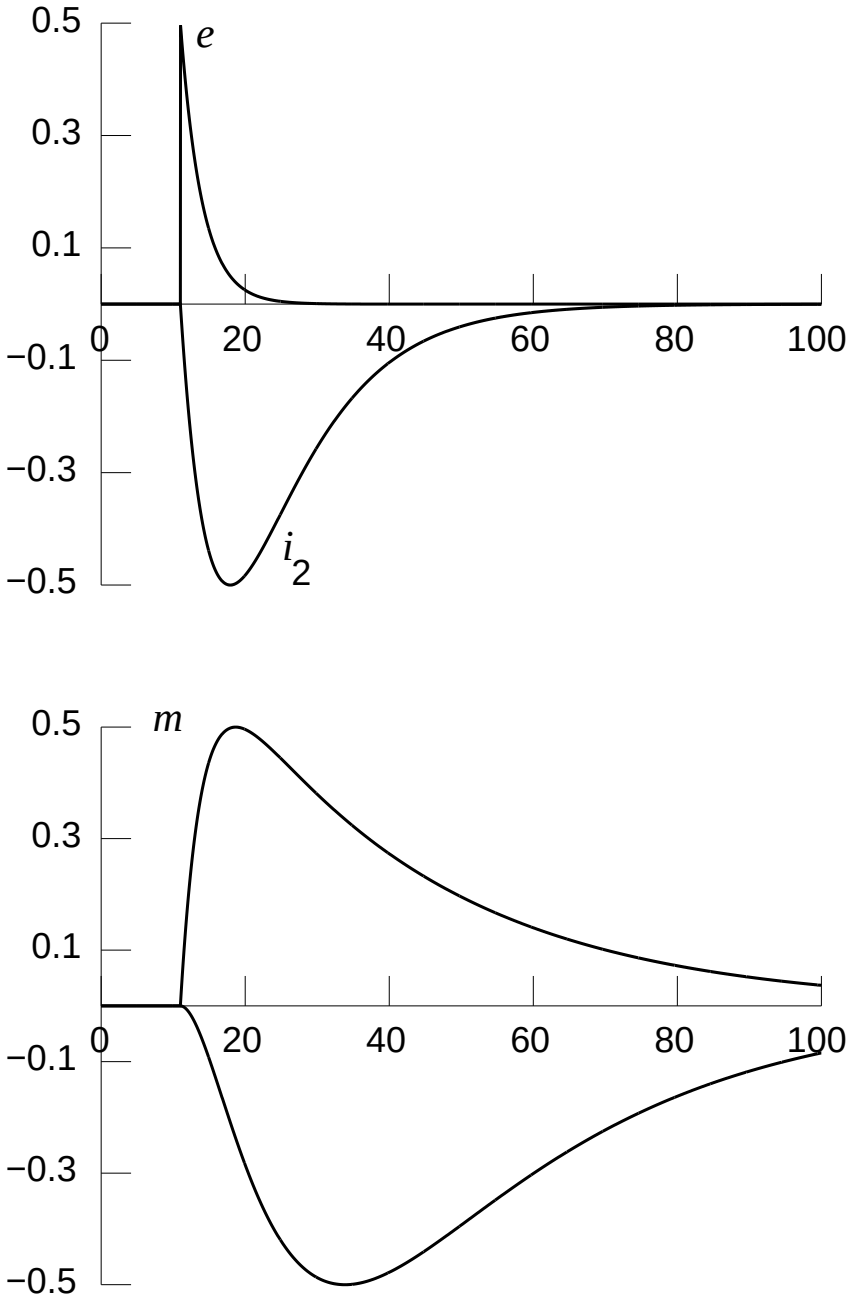
$$\infty \quad \text{if} \quad m'(t_0) \le 0$$

This is illustrated in **Fig. 6 top** where the arrow marks arrival of an external event. Although Eq. 13 predicts a finite t_f, this input is too weak to drive the cell to fire. The vertical lines indicate self−events, at which new Newton iterations are performed. If $m' < 0$ immediately after an input event, as in **Fig. 6 middle**, both t_f and the true firing time are infinite.

If instead $m(t_0)$ reaches threshold, the cell "fires," generating a `net_event` and setting m to 0. The self−event is then moved to an approximate firing time that is computed from Eq. 13 using the values assumed by m and m' immediately after the "spike." This is shown in **Fig. 6 bottom**, where the slope approximation after the excitatory event is not marked, but the response clearly crosses threshold (asterisk). Following the spike, m is reset to 0 but bounces back because of persistent excitatory current. This dies away without eliciting a second spike, even though t_f is finite (dashed line).

We use this approach for several reasons. First, t_f is never later than the true firing time. This stipulation is of central importance because the simulation would otherwise be in error. Second, successive approximations must converge rapidly to the true firing time, in order to avoid the overhead of a large number of self−events. Using the slope approximation to *m* is equivalent to the Newton method for solving $m(t) = 1$, so convergence is slow only when the maximum value of *m* is close to 1. Finally, the use of a series of self−events is superior to carrying out a complete Newton method solution because it is most likely that external input events will arrive in the interval between firing times. Each external event would invalidate the previous computation of firing time and force a recalculation. This might be acceptable for the IntFire2 mechanism with its efficient convergence, but the complicated dynamics of IntFire4 suggest that the cost would be too high. How many iterations should be carried out per self−event is an experimental question, since the self−event overhead depends partly on the number of outstanding events in the event queue.

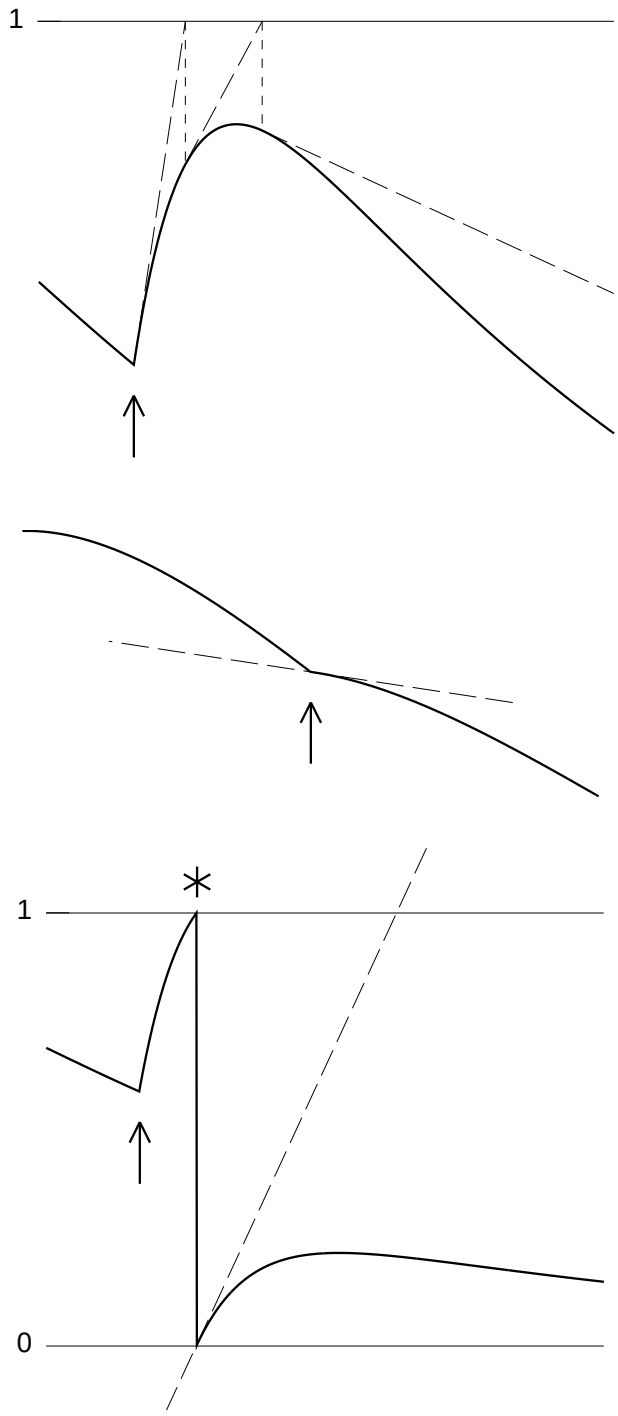
Figure 5



IntFire4 responses to individual events with w = 0.5 and −0.5.

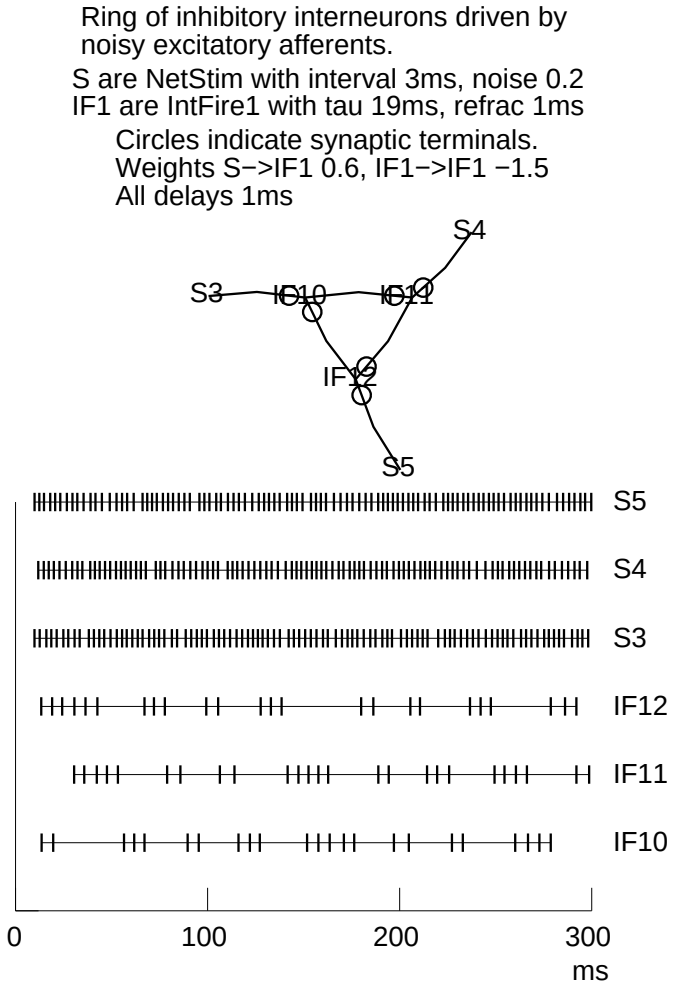
τ_e = 3 ms, τ_{i1} = 5 ms, τ_{i2} = 10 ms, τ_m = 30 ms

Figure 6



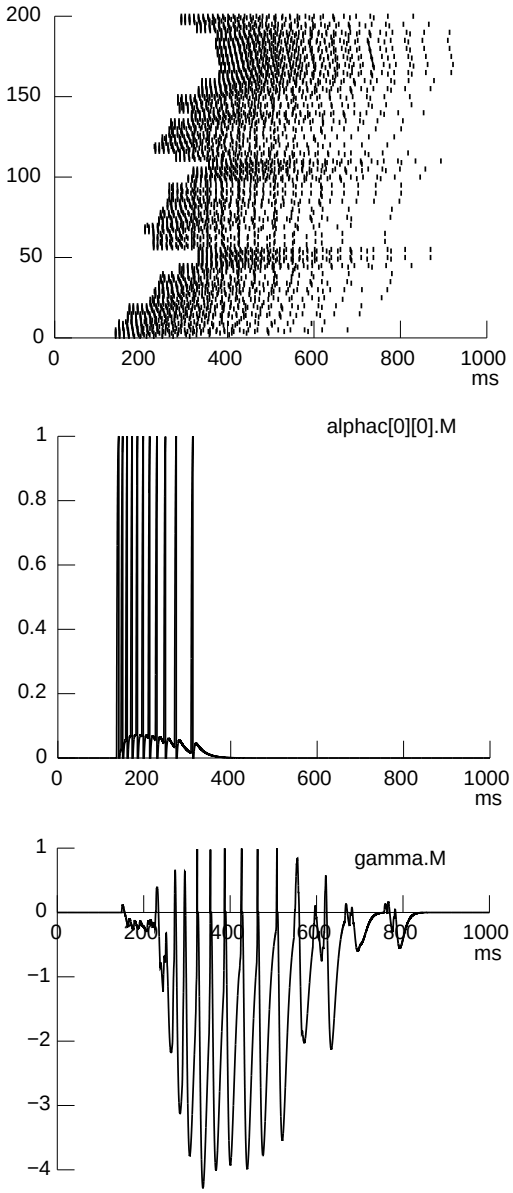
Examples using integrate and fire neurons

A ring of inhibitory neurons driven by noisy excitatory input can produce a noisy but cyclic firing pattern if the ring contains an odd number of neurons (Friesen and Friesen 1994). Executing the model shown here for 300,000 ms (>300,000 received events) required 30 seconds of runtime on a 2.2 GHz P4 with 512 MB RAM.



A scaled−down version of the Hopfield−Brody network (Hopfield and Brody 2000; 2001) was implemented with 401 IntFire4 cells that had been modified by including an absolute refractory interval. The figures below demonstrate typical activity of the trained net during a recognition task: afferent spike trains (top), *m* in an α cell (middle), and *m* in a γ cell (bottom). The particular simulation shown here took 16 seconds to run, during which there was a total of 250,970 received

events (63,785 self−events, 96,618 excitatory input events, and 90,567 inhibitory input events).



Discussion

NEURON's application domain now extends beyond continuous system simulations of models of individual neurons with complex anatomical and biophysical properties, to encompass discrete−event and hybrid simulations that combine "biological" and "artificial" neuronal models. The integrate−and−fire models presented here are distributed as part of the NEURON simulation environment, which is available from `http://www.neuron.yale.edu` at no charge.

Since NEURON's library of mechanisms is extensible through the NMODL programming language (Hines and Carnevale 2000), these mechanisms can be modified to add new features, and other formulations of artificial neuron models that have analytic solutions can be added by users as the need arises. For example, revisions of IntFire1 and 2 that implement short−term use−dependent plasticity as described by (Tsodyks et al. 2000) and (Varela et al. 1997) have already been made available at ModelDB `http://senselab.med.yale.edu/senselab/modeldb/`

Acknowledgment

This work was supported by: NIH grant NS011613.

References

Destexhe, A., Mainen, Z.F., and Sejnowski, T.J.. An efficient method for computing synaptic conductances based on a kinetic model of receptor binding. *Neural Computation* 6:14−18, 1994.

Destexhe, A., McCormick, D.A., and Sejnowski, T.J.. A model for 8−10 Hz spindling in interconnected thalamic relay and reticularis neurons. *Biophys. J.* 65:2474−2478, 1993.

Friesen, W.O. and Friesen, J.A.. *NeuroDynamix.* Oxford University Press, New York, 1994.

Hines, M.. A program for simulation of nerve equations with branching geometries. *Int. J. Bio−Med. Comput.* 24:55−68, 1989.

Hines, M.. NEURON−−a program for simulation of nerve equations, in *Neural Systems: Analysis and Modeling*, ed. F. Eeckman, Kluwer, Norwell, MA, 1993, pp. 127−136.

Hines, M. and Carnevale, N.T.. Computer modeling methods for neurons, in *The Handbook of Brain Theory and Neural Networks* edition 1, ed. M.A. Arbib, MIT Press, Cambridge, MA, 1995, pp. 226−230.

Hines, M.L. and Carnevale, N.T.. Expanding NEURON's repertoire of mechanisms with NMODL. *Neural Computation* 12:839−851, 2000.

Hopfield, J.J. and Brody, C.D.. What is a moment? "Cortical" sensory integration over a brief interval. *PNAS* 97:13919−13924, 2000.

Hopfield, J.J. and Brody, C.D.. What is a moment? Transient synchrony as a collective mechanism for spatiotemporal integration. *PNAS* 98:1282−1287, 2001.

Lytton, W.W.. Optimizing synaptic conductance calculation for network simulations. *Neural Computation* 8:501−509, 1996.

Lytton, W.W., Contreras, D., Destexhe, A., and Steriade, M.. Dynamic interactions determine partial thalamic quiescence in a computer network model of spike−and−wave seizures. *J. Neurophysiol.* 77:1679−1696, 1997.

Maas, W. and Bishop, C.M., eds.. *Pulsed Neural Networks*. MIT Press, Cambridge, MA, 1999.

Riecke, F., Warland, D., de Ruyter van Steveninck, R., and Bialek, W.. *Spikes: Exploring the Neural Code*. MIT Press, Cambridge, MA, 1997.

Sohal, V.S., Huntsman, M.M., and Huguenard, J.R.. Reciprocal inhibitory connections regulate the spatiotemporal properties of intrathalamic oscillations. *J. Neurosci.* 20:1735−1745, 2000.

Tsodyks, M., Uziel, A., and Markram, H.. Synchrony generation in recurrent networks with frequency−dependent synapses. *J. Neurosci.* 20:1−5, 2000.

Varela, J.A., Sen, K., Gibson, J., Fost, J., Abbott, L.F., and Nelson, S.B.. A quantitative description of short−term plasticity at excitatory synapses in layer 2/3 of rat primary visual cortex. *J. Neurosci.* 17:7926−7940, 1997.