An introduction to NineML

Ivan Raikov

Okinawa Institute of Science and Technology
Computational Neuroscience Unit

September 5, 2011

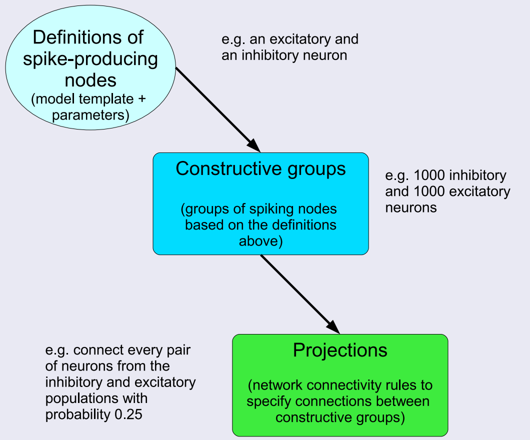
incf Program on Multi-Scale Modeling

Background

- NineML is a domain-specific language for describing neuronal network models.
- Joint effort of developers of neuroscience simulator software and simulator-independent description language initiatives (NeuroML, PyNN).
- The NineML effort is initiated and coordinated by the International Neuroinformatics Coordinating Facility (INCF).

Model structure of NineML

Model



A model description consists of:

- Neuronal dynamics template
- List of parameter values
- Groups of neurons–populations, layers, etc.
- Connectivity patterns–projections, connection probabilities, etc.

Spike-producing node definition

Example

The user starts out by choosing a model template:

Model template: `IaF_gL`
(supplied by model library)

$$C_m \frac{dV}{dt} = -g_L V + \sum_i I_i$$

$$\text{if } V > \theta : \begin{cases} \text{while } t < t_{spike} + t_{refr} \\ V = V_{rest} \end{cases}$$

Having chosen a model template, the user can then provide parameter values:

Parameter description
(supplied by user)

$C_m = 1\mu F$ $\quad g_L = 0.5mS$

$\theta = 20mV$ $\quad V_{rest} = -60mV$

$t_{refr} = 0.1ms$

Spike-producing node definition

Example

Model template: `IaF_gL`
(supplied by model library)

$$C_m \frac{dV}{dt} = -g_L V + \sum_i I_i$$

$$\text{if } V > \theta : \begin{cases} \text{while } t < t_{spike} + t_{refr} \\ V = V_{rest} \end{cases}$$

Parameter description
(supplied by user)

$C_m = 1\mu F$ $\quad$ $g_L = 0.5mS$

$\theta = 20mV$ $\quad$ $V_{rest} = -60mV$

$t_{refr} = 0.1ms$

Spike-producing node definition

Example

```
<spikingNode name="E">

<definition>
  <type>IaF_gL</type>
  <link><url>http://www.nineml.org/1.0/IaF_gL</url></link>
</definition>

<parameters>
  <membraneCapacitance name="C_m">...
    <value>1</value>
    <unit>uF</unit>
  </membraneCapacitance>
  <threshold>...</threshold>
  <refractoryTime>...</refractoryTime>
  <resetPotential>...</resetPotential>
</parameters>
```

Spike-producing node definition

Example

```
<spikingNode name="E">

<definition>
  <type>IaF_gL</type>
  <link><url>http://www.nineml.org/1.0/IaF_gL</url></link>
</definition>

<parameters>
  <membraneCapacitance name="C_m">...
    <value>1</value>
    <unit>uF</unit>
  </membraneCapacitance>
  <threshold>...</threshold>
  <refractoryTime>...</refractoryTime>
  <resetPotential>...</resetPotential>
</parameters>
```

Spike-producing node definition

Example

```
<spikingNode name="E">

<definition>
  <type>IaF_gL</type>
  <link><url>http://www.nineml.org/1.0/IaF_gL</url></link>
</definition>
<parameters>

  <membraneCapacitance name="C_m">...
    <value>1</value>
    <unit>uF</unit>
  </membraneCapacitance>

  <threshold>...</threshold>
  <refractoryTime>...</refractoryTime>
  <resetPotential>...</resetPotential>
</parameters>
```

User Layer and Abstraction Layer

User Layer

Parameter description
(supplied by user)

$$C_m = 1\mu F \qquad g_L = 0.5mS$$

$$\theta = 20mV \qquad V_{rest} = -60mV$$

$$t_{refr} = 0.1ms$$

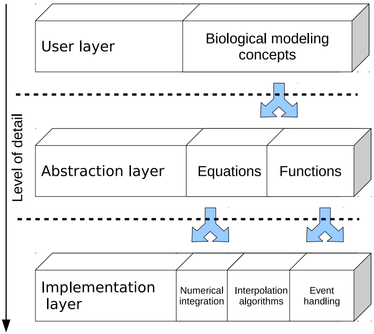
Abstraction Layer

Model template: `IaF_gL`
(supplied by model library)

$$C_m \frac{dV}{dt} = -g_L V + \sum_i I_i$$

$$\text{if } V > \theta : \begin{cases} \text{while } t < t_{spike} + t_{refr} \\ V = V_{rest} \end{cases}$$

The Layer View



Parameters for model `IaF_gL`

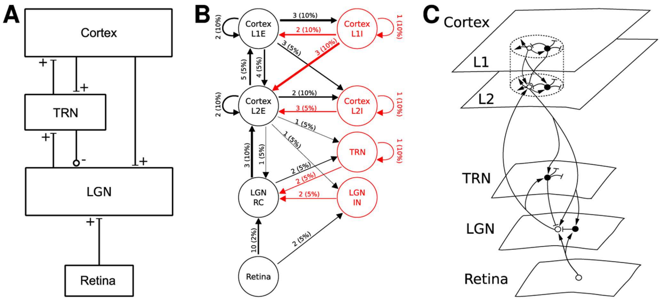
$$C_m = 1\mu F \qquad g_L = 0.5mS$$
$$\theta = 20mV \qquad V_{rest} = -60mV$$
$$t_{refr} = 0.1ms$$

Model template: `IaF_gL`

$$C_m \frac{dV}{dt} = -g_L V + \sum_i I_i$$
$$\text{if } V > \theta : \begin{cases} \text{while } t < t_{spike} + t_{refr} \\ V = V_{rest} \end{cases}$$

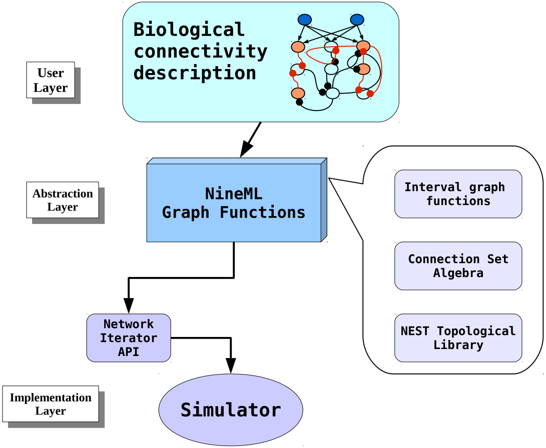
- In the user layer, we characterize biological concepts by means of a model name and a set of parameter values.
- In the abstraction layer, we define parameterized equations that characterize a particular class of models.

Representations of Network Connectivity

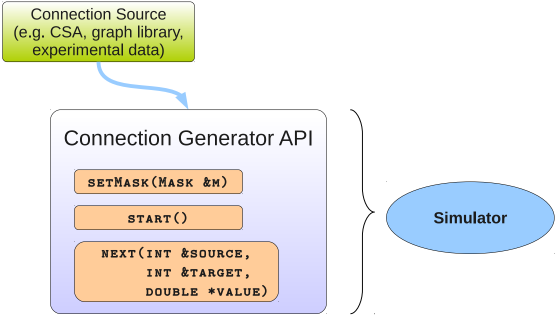


Nordlie, E et al. *Towards reproducible descriptions of neuronal network models.*, PLoS Comput Biol (2009).

NineML Connectivity Overview



Connection Generator Interface



Masks and Interval Sets

```
class Mask
{
IntervalSet sources;
IntervalSet targets;
}
```

```
class IntervalSet
{
std::vector<ClosedInterval> ivals;

}
```

Network Connectivity Example

Create index intervals for excitatory, inhibitory and all cells

```
binding e = Interval.closed_interval (0, 599)

binding i = Interval.closed_interval (600, 899)

binding a = Interval.union (e, i)
```

Network Connectivity Example

Create geometry function g and metric d

```
binding g = CSA.random2d (900)

binding d = CSA.euclidMetric2d (g)
```

Network Connectivity Example

Excitatory and inhibitory gaussian value sets (gaussian of the distance for every index pair)

```
binding g_e = CSA.gaussian (0.1, 0.3) (d)

binding g_i = CSA.gaussian (0.2, 0.3) (d)
```

Network Connectivity Example

Create connection-sets with gaussian dependent random masks, gaussian dependent conductance and distance dependent delay

```
binding c_e = CSA.cset (CSA.random (g_e), g_e, d)

binding c_i = CSA.cset (CSA.random (g_i), -g_i, d)
```

Network Connectivity Example

Combine excitatory and inhibitory connectivity into one network using cartesian and multiset sum operators

```
binding c = CSA.sum (CSA.cartesian (e, a) (c_e),
                     CSA.cartesian (i, a) (c_i))
```

Network Connectivity Example

Create index intervals for excitatory, inhibitory and all cells

```
binding e = Interval.closed_interval (0, 599)
binding i = Interval.closed_interval (600, 899)
binding a = Interval.union (e, i)
```

Create geometry function g and metric d

```
binding g = CSA.random2d (900)
binding d = CSA.euclidMetric2d (g)
```

Excitatory and inhibitory gaussian value sets (gaussian of the distance for every index pair)

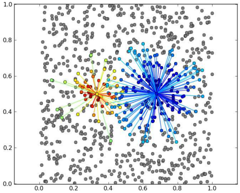
```
binding g_e = CSA.gaussian (0.1, 0.3) (d)
binding g_i = CSA.gaussian (0.2, 0.3) (d)
```

Create connection-sets with gaussian dependent random masks, gaussian dependent conductance and distance dependent delay

```
binding c_e = CSA.cset (CSA.random (g_e), g_e, d)
binding c_i = CSA.cset (CSA.random (g_i), -g_i, d)
```

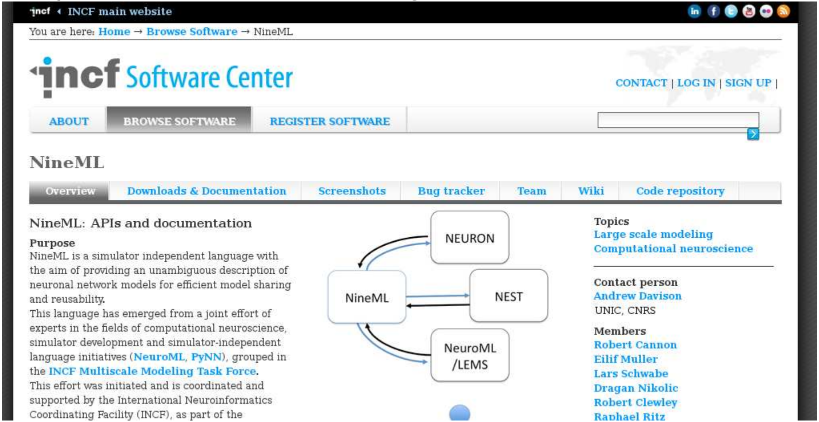
Combine excitatory and inhibitory connectivity into one network using cartesian and multiset sum operators

```
binding c = CSA.sum (CSA.cartesian (e, a) (c_e),
                     CSA.cartesian (i, a) (c_i))
```



Home page

http://software.incf.org/software/nineml



Contributors

Abigail Morrison, RIKEN Brain Science Institute;

Andrew Davison, CNRS;

Botond Szatmary, Brain Corporation;

Eilif Muller, LCN, EPFL;

Hans Ekkehard Plesser, Norwegian University of Life Sciences;

Lars Schwabe, University of Rostock;

Mikael Djurfeldt, INCF;

Robert Cannon, Textensor Limited;

Sean Hill, INCF;

Yann Le Franc, University of Antwerp;

Anatoli Gorchetchnikov, Boston University;

Birgit Kriener, Max Planck Inst. for Dynamics and Self-Organization;

Chung-Chan Lo, National Tsing Hua University;

Erik De Schutter, Okinawa Institute of Science and Technology;

Hugo Cornelis, University of Texas Health Science Center;

Michael Hines, Yale University;

Padraig Gleeson, University College London;

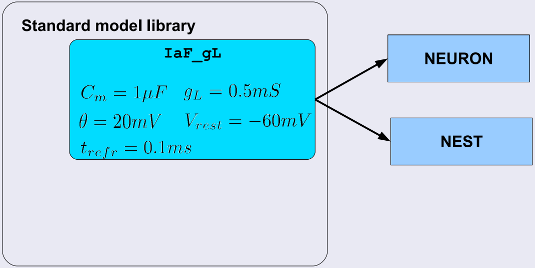
Robert Clewley, Georgia State University;

Subhasis Ray, National Center for Biological Sciences;

Why the Need for Layers?

Let us consider a non-layered solution, e.g. PyNN or NeuroML:

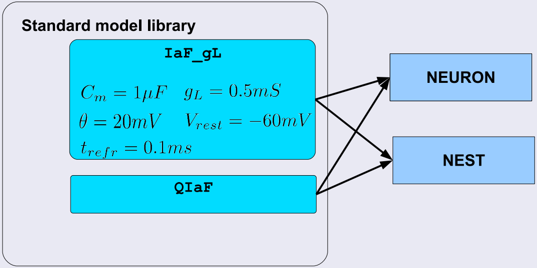
A non-layered solution for model standardization



Why the Need for Layers?

Let us consider a non-layered solution, e.g. PyNN or NeuroML:

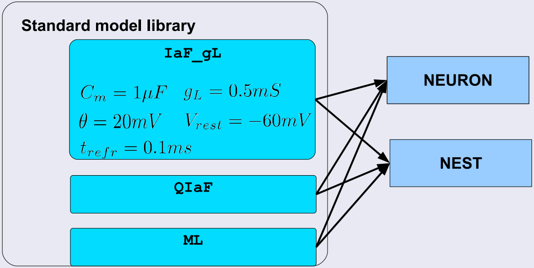
A non-layered solution for model standardization



Why the Need for Layers?

Let us consider a non-layered solution, e.g. PyNN or NeuroML:

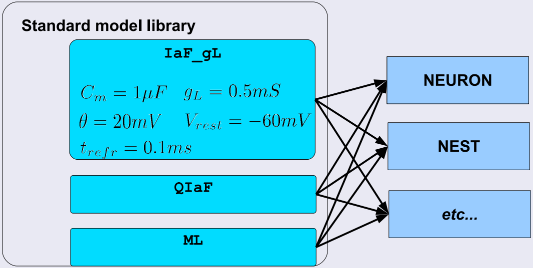
A non-layered solution for model standardization



Why the Need for Layers?

Let us consider a non-layered solution, e.g. PyNN or NeuroML:

A non-layered solution for model standardization



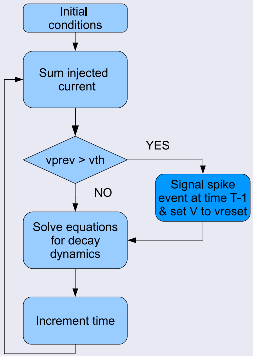
Problems with the non-layered approach:

- Potential duplication of implementation effort.
- Superficial model equivalence.
- Insufficiently flexible for development of new models.

Why the Need for Layers?

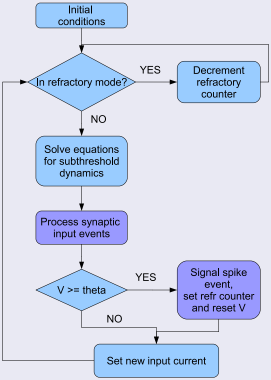
The problem of porting models between simulators:

Generalized Integrate-and-Fire



Pillow, JW, Paninski, L., et al. (2005): Prediction and Decoding of Retinal Ganglion Cell Responses with a Probabilistic Spiking Model. *Journal of Neuroscience* 25:11003-11013.

NEST extension module

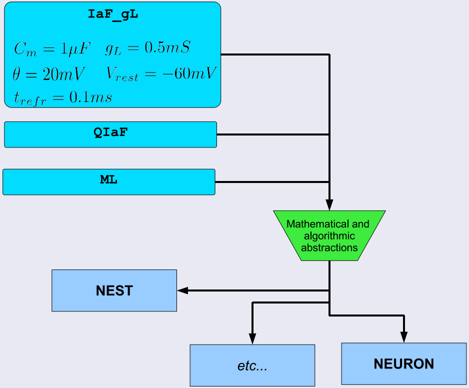


Writing an Extension Module with NEST.
`http://www.nest-initiative.org/index.php/Writing_an_Extension_Module`

Why the Need for Layers?

The answer: flexibility, ease of porting models, establishing true model equivalence.

A layered solution for model standardization



We've Got Layers, Now What?

- What should the abstraction layer for networks of integrate-and-fire models look like?

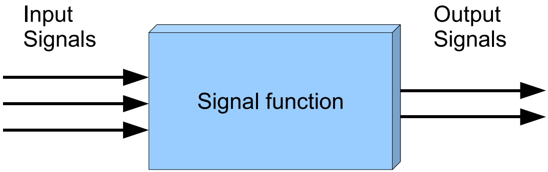
We've Got Layers, Now What?

- What should the abstraction layer for networks of integrate-and-fire models look like?
- Such an abstraction layer must:
  - provide enough flexibility for a variety of existing models
  - capture mathematical and algorithmic model properties required to achieve reproducibility of results
  - provide extensibility for future model development

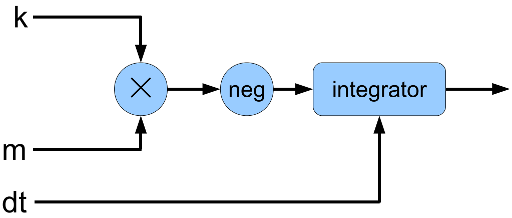
We've Got Layers, Now What?

- What should the abstraction layer for networks of integrate-and-fire models look like?
- Such an abstraction layer must:
  - provide enough flexibility for a variety of existing models
  - capture mathematical and algorithmic model properties required to achieve reproducibility of results
  - provide extensibility for future model development
- Proposal: an abstraction layer that can represent:
  - event handling
  - state transitions
  - differential and algebraic equations
  - other equation types: chemical reaction equations, etc.

Signal Functions

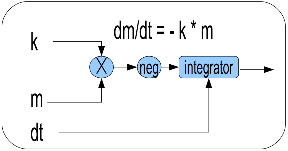


dm/dt = - k * m

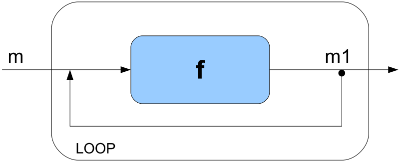


Signal Function Combinators

f =

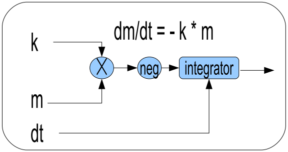


LOOP [(m m1)] f

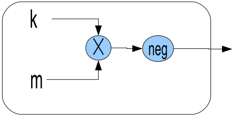


Signal Function Combinators

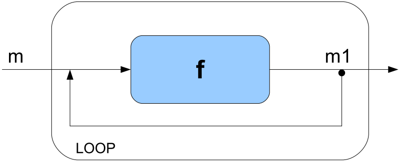
f =



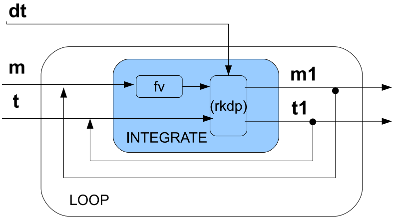
fv =



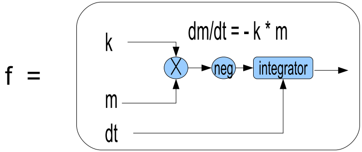
LOOP [(m m1)] f



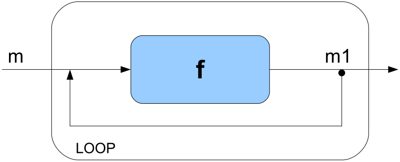
LOOP [(m m1) (t t1)]
INTEGRATE rkdp [(m m1) (t t1) dt] fv



Signal Function Combinators



LOOP [(m m1)] f



Advantages of signal function combinators:

- precise control over the structure of the abstraction layer
- much flexibility in building the mathematical description of a model
- efficient executable code from model specification

Signal Function Combinators: Example 1

```
DEFINE dv (v,w,Istim,gl,vl,v1,v2,gca,vca,gk,vk,c)
    (Istim + (gl * (vl - v)) +
             ica(v,gca,vca,v1,v2) +
             ik(v,w,gk,vk)) / c

DEFINE dw (w,v,phi,v3,v4)
    lamw (v,phi,v3,v4) * (winf (v,v3,v4) - w)


DEFINE Morris-Lecar
 LOOP [(v nv) (w nw) (t nt)]
   SEQ
     INTEGRATE [(v nv) (t nt) tstep]
      SENSE [v w Istim gl vl v1 v2 gca vca gk vk c]
        PURE (dv) dv
    INTEGRATE [(w nw) (t nt) tstep]
      SENSE (w v phi v3 v4)
        PURE (dw) dw
```

Signal Function Combinators: Example 1

```
DEFINE dv (v,w,Istim,gl,vl,v1,v2,gca,vca,gk,vk,c)
    (Istim + (gl * (vl - v)) +
             ica(v,gca,vca,v1,v2) +
             ik(v,w,gk,vk)) / c

DEFINE dw (w,v,phi,v3,v4)
    lamw (v,phi,v3,v4) * (winf (v,v3,v4) - w)


DEFINE Morris-Lecar
  LOOP [(v nv) (w nw) (t nt)]
    SEQ

    INTEGRATE [(v nv) (t nt) tstep]
     SENSE (v w Istim gl vl v1 v2 gca vca gk vk c)
       PURE (dv) dv

   INTEGRATE [(w nw) (t nt) tstep]
     SENSE (w v phi v3 v4)
       PURE (dw) dw
```

Signal Function Combinators: Example 1

```
DEFINE dv (v,w,Istim,gl,vl,v1,v2,gca,vca,gk,vk,c)
    (Istim + (gl * (vl − v)) +
             ica(v,gca,vca,v1,v2) +
             ik(v,w,gk,vk)) / c

DEFINE dw (w,v,phi,v3,v4)
    lamw (v,phi,v3,v4) * (winf (v,v3,v4) − w)


DEFINE Morris−Lecar
 LOOP [(v nv) (w nw) (t nt)]
   SEQ

    INTEGRATE [(v nv) (t nt) tstep]
     SENSE (v w Istim gl vl v1 v2 gca vca gk vk c)
       PURE (dv) dv

   INTEGRATE [(w nw) (t nt) tstep]
     SENSE (w v phi v3 v4)
       PURE (dw) dw
```

Signal Function Combinators: Example 2

```
DEFINE Hansel:98
  LOOP [(v nv) (t nt)]
    RSWITCH [spike! recur]
      SEQ
          INTEGRATE rkdp [(v nv) (t nt) tstep]
            SENSE [v t spikes Istim c vl gl gsyn ve tau1 tau2]
              PURE [dv] dv
          ACTUATE [spike!]
            SENSE [nv nt theta]
              PURE [spike!] spike?
      ACTUATE [v recur]
        SENSE [Vrest]
          PURE/list [recur] refr
```

Signal Function Combinators: Example 2

```
DEFINE Hansel:98
  LOOP [(v nv) (t nt)]

    RSWITCH [spike! recur]

      SEQ
         INTEGRATE rkdp [(v nv) (t nt) tstep]
           SENSE [v t spikes Istim c vl gl gsyn ve tau1 tau2]
             PURE [dv] dv
         ACTUATE [spike!]
           SENSE [nv nt theta]
             PURE [spike!] spike?
      ACTUATE [v recur]
        SENSE [Vrest]
          PURE/list [recur] refr
```

Signal Function Combinators: Example 2

```
DEFINE Hansel:98
  LOOP [(v nv) (t nt)]
    RSWITCH [spike! recur]

      SEQ
         INTEGRATE rkdp [(v nv) (t nt) tstep]
           SENSE [v t spikes Istim c vl gl gsyn ve tau1 tau2]
             PURE [dv] dv
         ACTUATE [spike!]
           SENSE [nv nt theta]
             PURE [spike!] spike?

      ACTUATE [v recur]
        SENSE [Vrest]
          PURE/list [recur] refr
```

Signal Function Combinators: Example 2

```
DEFINE Hansel:98
  LOOP [(v nv) (t nt)]
    RSWITCH [spike! recur]
      SEQ
         INTEGRATE rkdp [(v nv) (t nt) tstep]
           SENSE [v t spikes Istim c vl gl gsyn ve tau1 tau2]
             PURE [dv] dv
         ACTUATE [spike!]
           SENSE [nv nt theta]
             PURE [spike!] spike?

      ACTUATE [v recur]
        SENSE [Vrest]
          PURE/list [recur] refr
```

Signal Function Combinators: Example 2

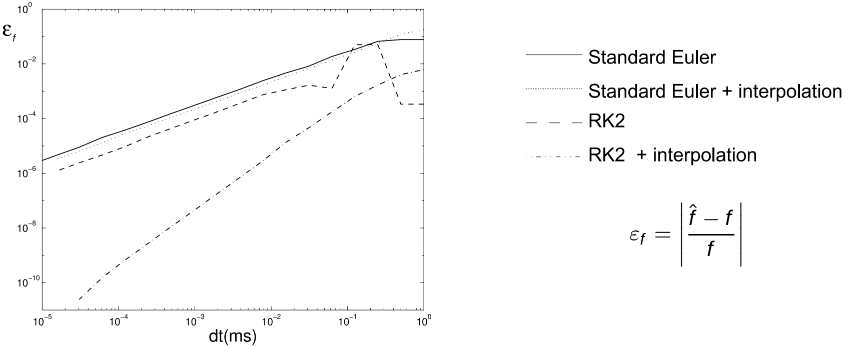
```
DEFINE Isyn (v,t,spikes,gsyn,ve,tau1 tau2)
  (gsyn − (v − ve))  ∗
  (vector−fold Isynf 0.0 spikes)

DEFINE dv (v,t,spikes,Istim ,c,vl,gl,gsyn,ve,tau1,tau2)
  (((neg gl) ∗ (− v vl)) + Isyn (v,t,spikes,gsyn,ve,tau1,tau2)) / c

DEFINE refr (Vrest)
  (list Vrest 1)

DEFINE spike? (v,t,theta)
  if (>= v theta)
  then t
  else  undefined−signal
```

Numerical Issues



$$\varepsilon_f = \left| \frac{\hat{f} - f}{f} \right|$$

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