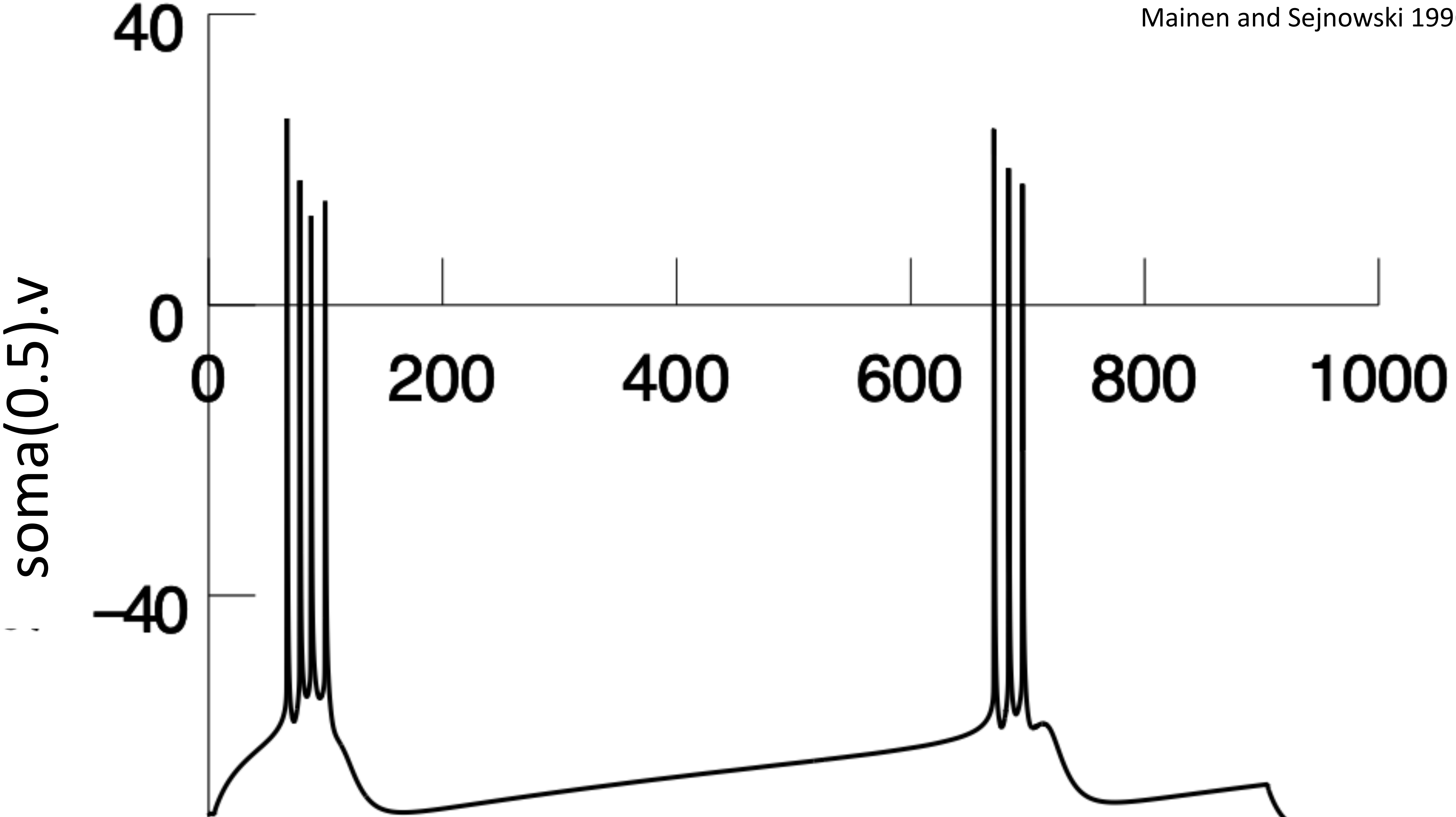
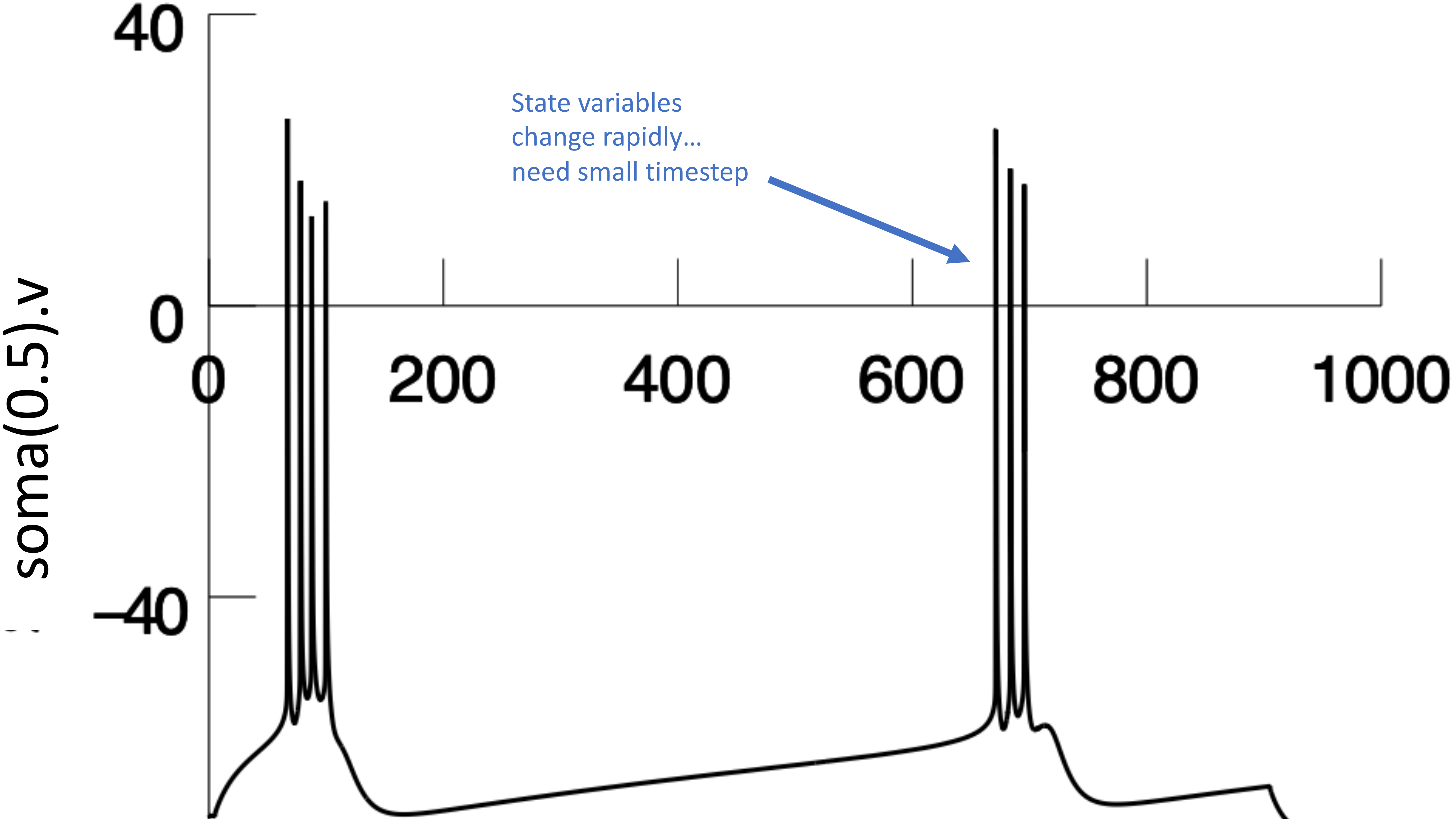
Numerical Methods:

Adaptive Integration

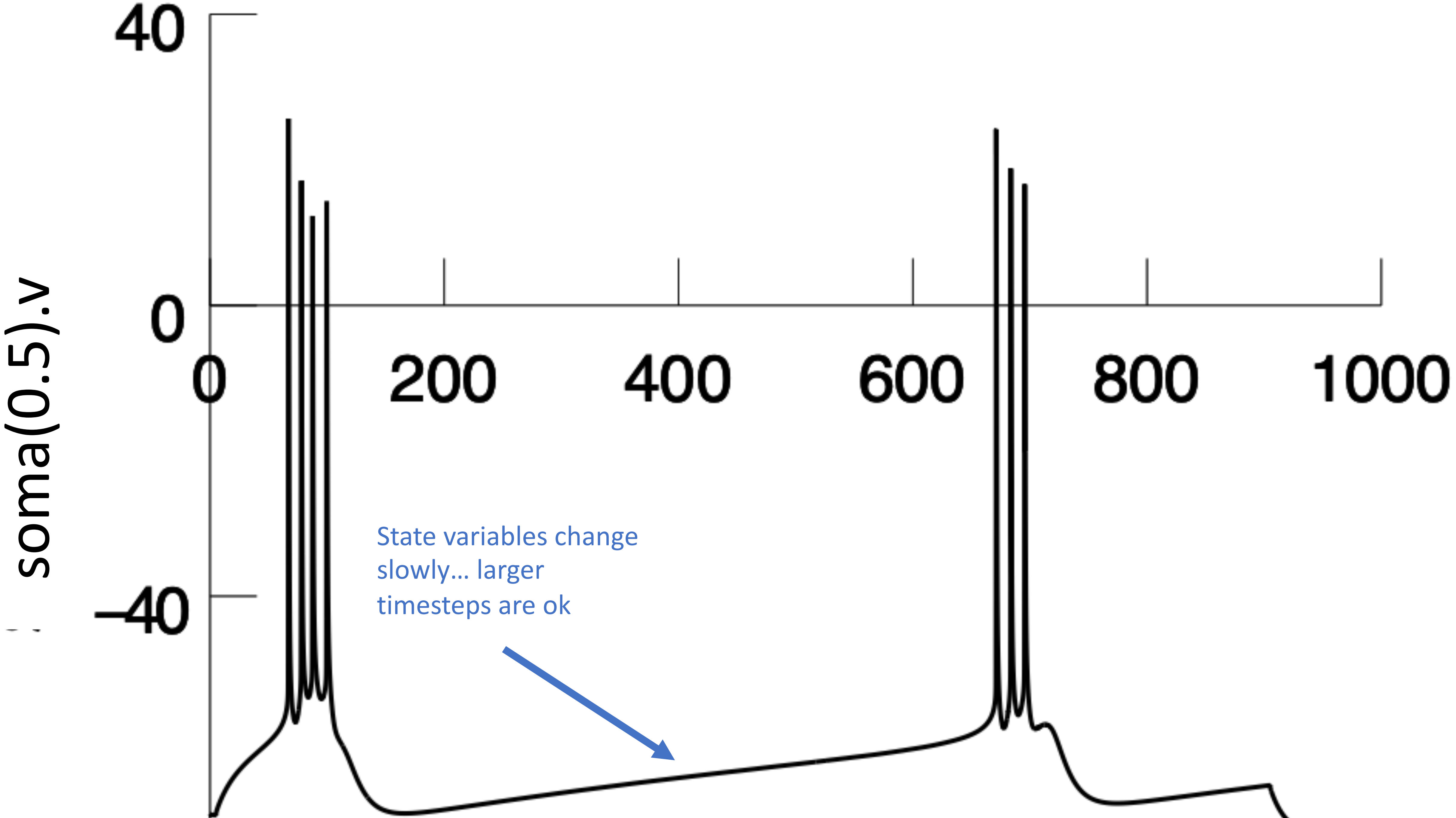
Based on model of https://modeldb.yale.edu/2488
Mainen and Sejnowski 1996



Based on model of https://modeldb.yale.edu/2488



Based on model of https://modeldb.yale.edu/2488



We’d like to use big timesteps when things are changing slowly, and small timesteps when things are changing quickly.

But we don’t know in advance when either will happen, so what do we do?

Math!

We’d like to use big timesteps when things are changing slowly, and small timesteps when things are changing quickly.

But we don’t know in advance when either will happen, so what do we do?

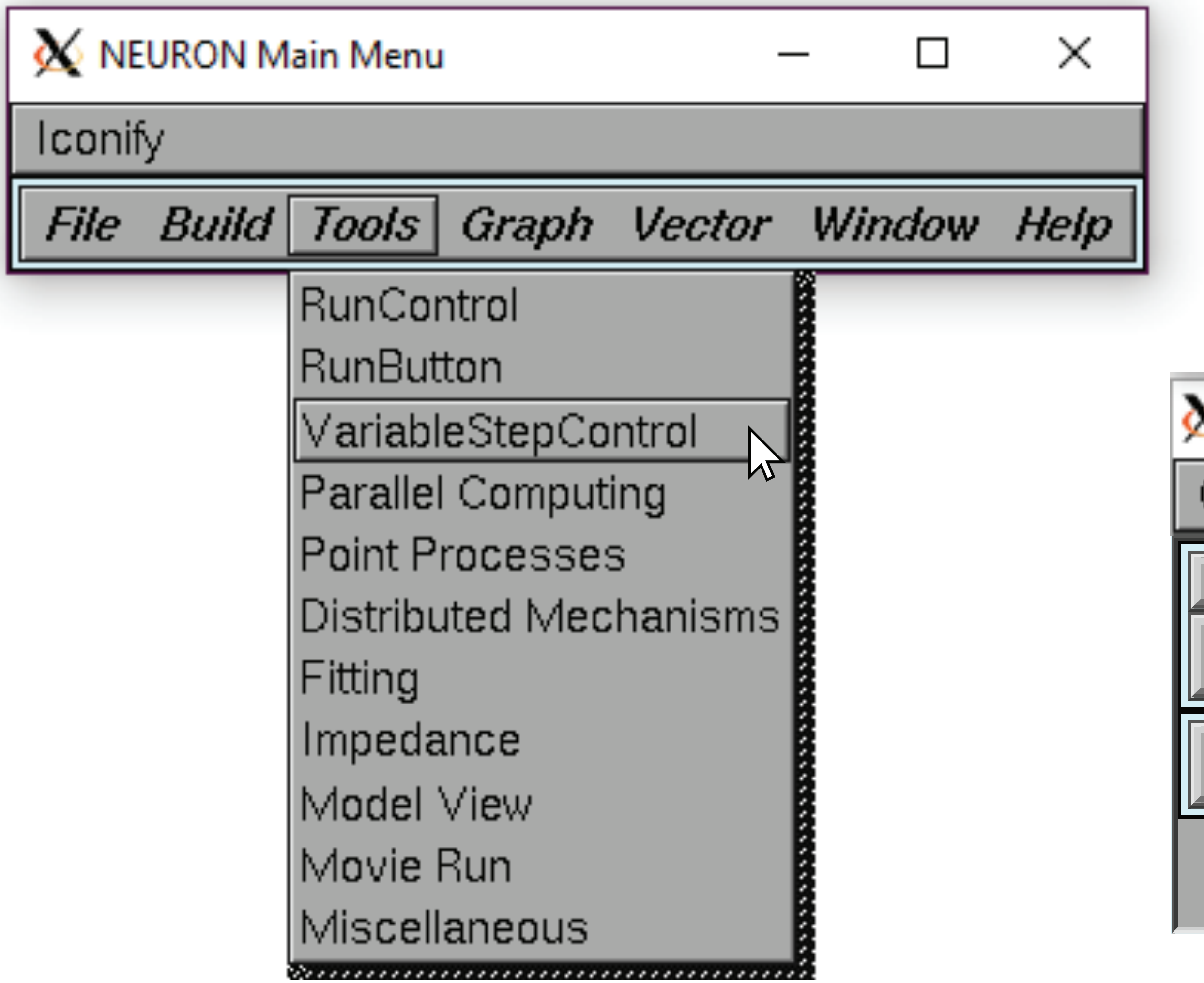
Variable step integration

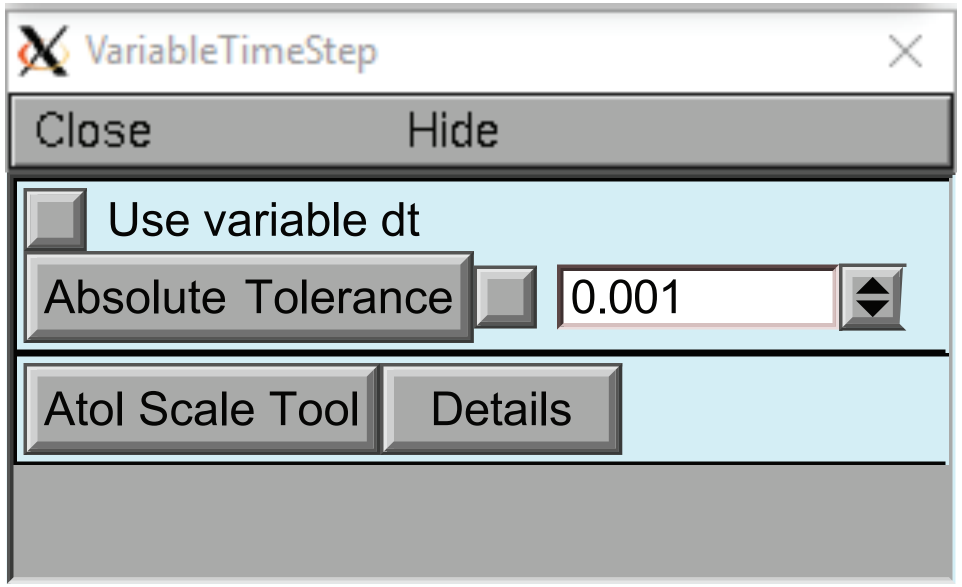
Enabling adaptive integration

`h.cvode_active(True)`

or

`h.CVode().active(True)`

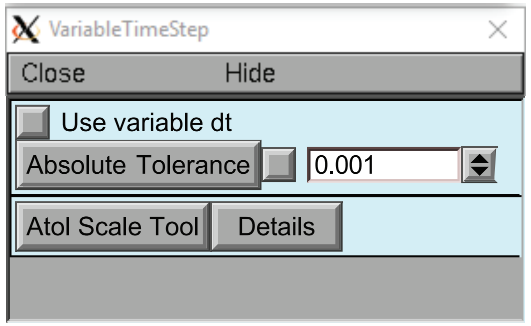


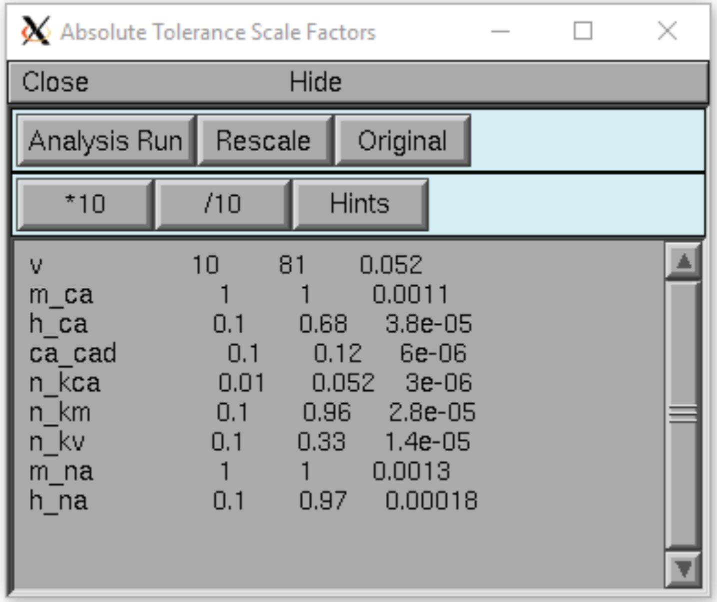


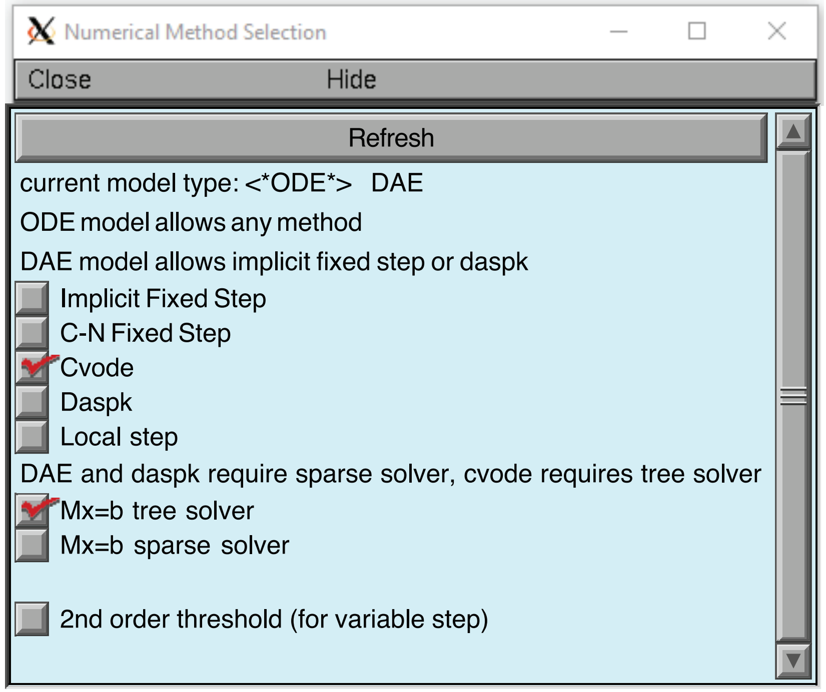
h.cvode active is defined in stdrun.hoc which is loaded automatically whenever the gui is imported.
This is a composite image, not a screenshot to allow combining the window decoration and vector-based window contents.

Options: per state variable tolerance, integration methods

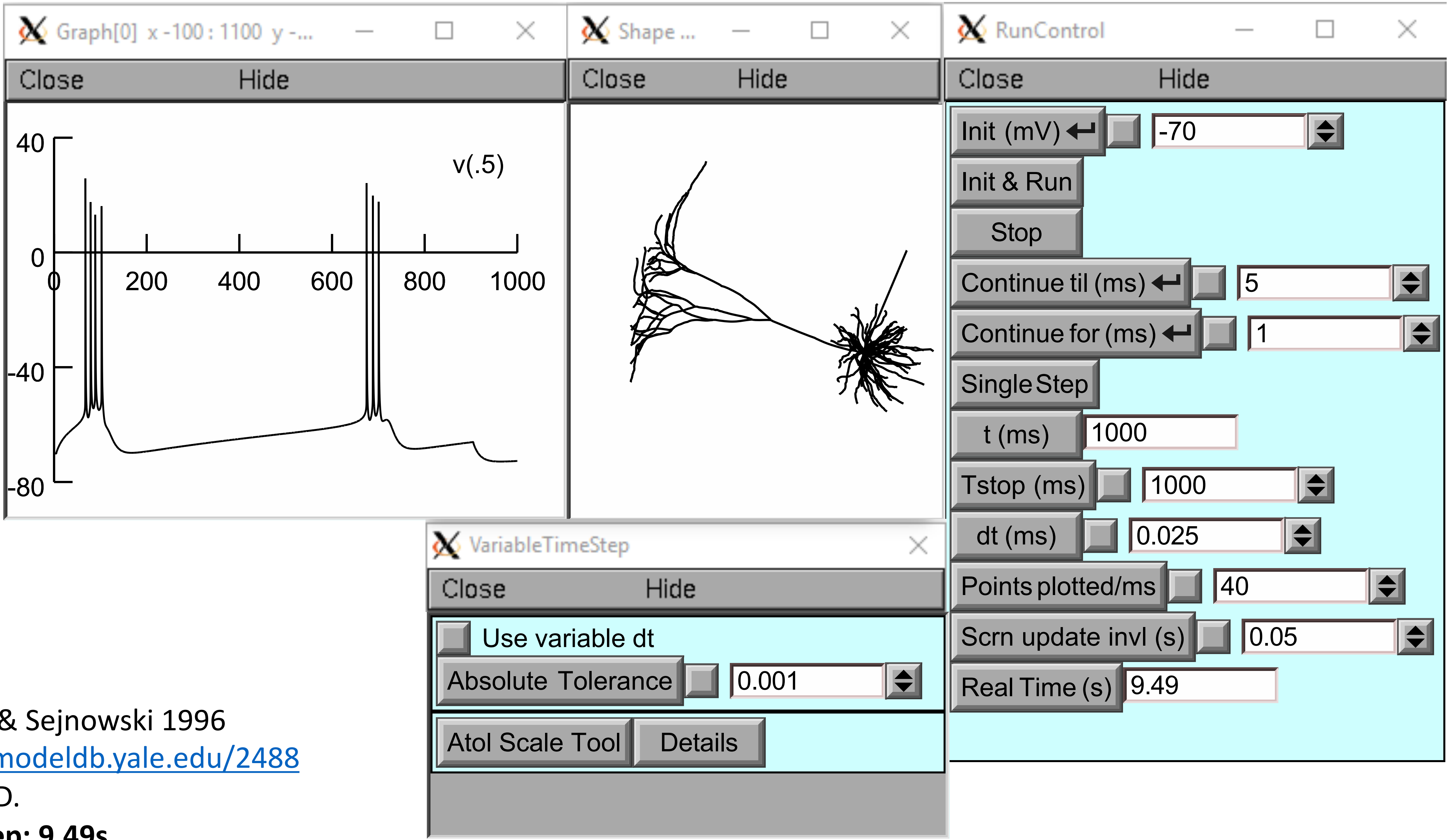
```
h.CVode().atol(0.001)
```



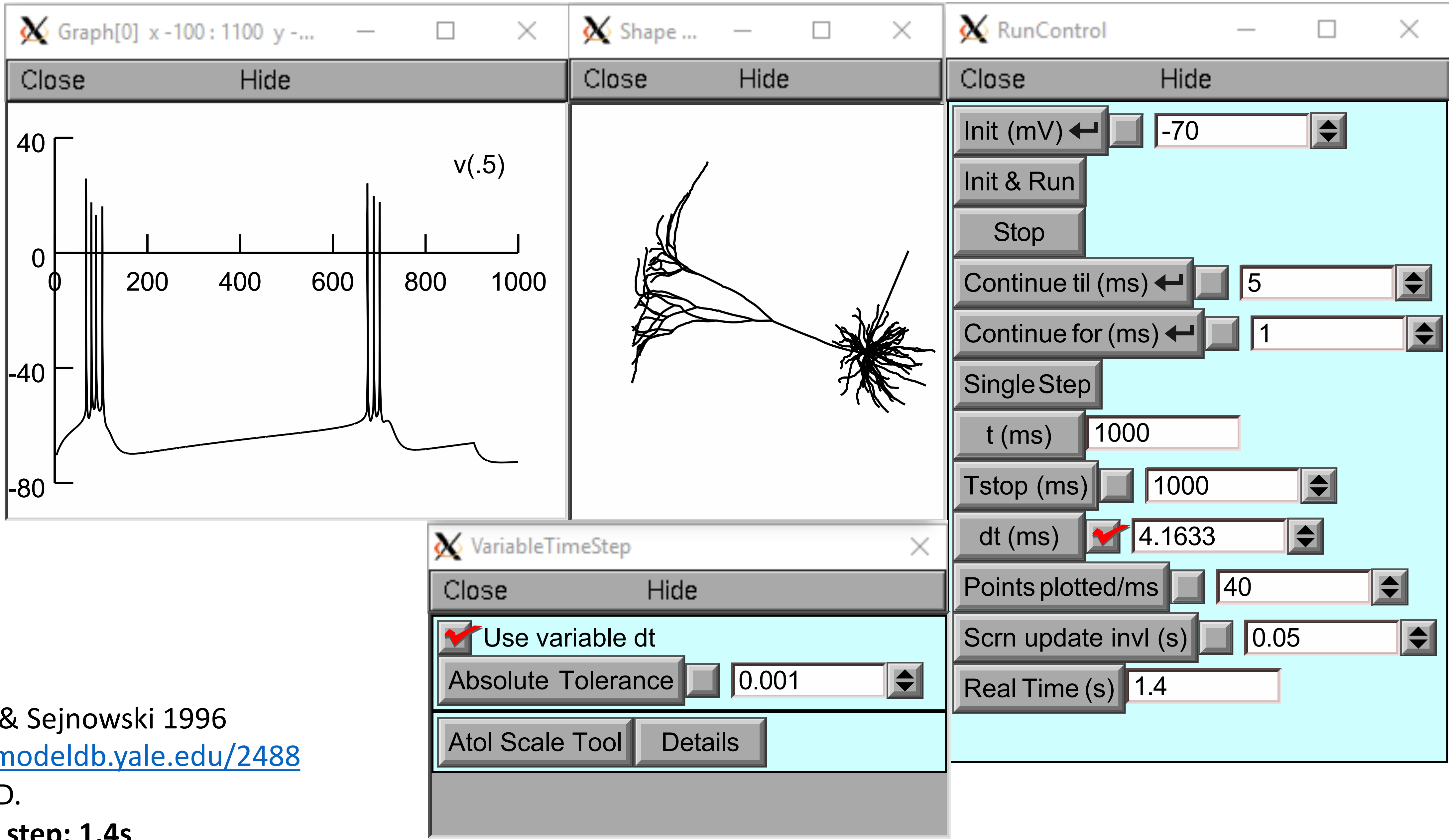




This is a composite image, not a screenshot to allow combining the window decoration and vector-based window contents.



Mainen & Sejnowski 1996
https://modeldb.yale.edu/2488
Figure 1D.
Fixed step: 9.49s
NEURON 7.5 on a 3.4 GHz i7-4770 with 24 GB RAM in WSL • composite image



Mainen & Sejnowski 1996

https://modeldb.yale.edu/2488

Figure 1D.

Variable step: 1.4s

NEURON 7.5 on a 3.4 GHz i7-4770 with 24 GB RAM in WSL • composite image

Membrane

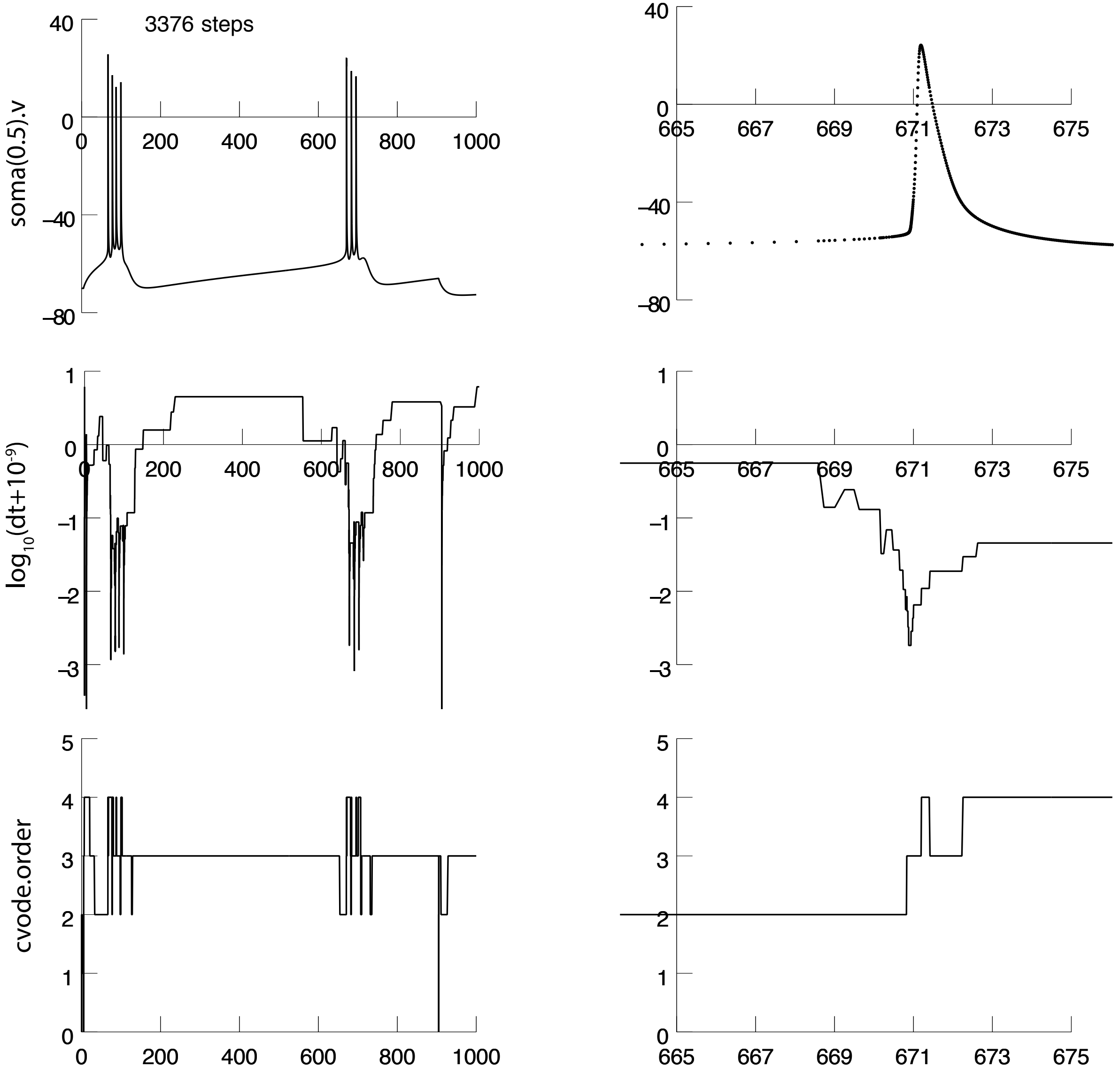
Potential

Timestep

Size

Order of

Integration Method



Running simulations: improving accuracy

Increase time resolution (by reducing time steps) via, e.g.

```
h.dt = 0.01 * ms
```

Enable variable step (allows error control):

```
h.CVode().active(True)
```

Set the absolute tolerance to e.g. 10^{-5}:

```
h.CVode().atol(1e-5)
```

Increase spatial resolution by e.g. a factor of 3 everywhere:

```
for sec in h.allsec(): sec.nseg *= 3
```