The leaky integrate and fire model

The leaky integrate and fire model is a simple model of the neuron; it has been used in neuroscience to study large networks of neurons and to investigate how neuronal networks might learn through changes in the strengths of synapses. Here we will look at a single neuron.

Say V is the voltage inside the neuron, the equation for how the voltage changes is given by

$$\tau_m \frac{dV}{dt} = E_L - V + R_m I \tag{1}$$

So

- dV/dt the rate of change of the voltage.
- I the current coming in to the cell changing the voltage, R_m basically measures how much the charge changes the voltage.
- $E_L - V$ the charge also leaks out and it leaks out more for bigger V. $E_L = -70$ mV, the reason this isn't zero is complicated and related to entropy.
- τ_m is a time constant, usually about 10 ms. It determines how fast V responds.

That's part of the behavior of the neuron; there is also spiking, that isn't modelled by the integrate and fire model. Instead there is a rule that when $V \geq V_T = -55$ mV there's a spike and the voltage is set back to zero.

Simulating an integrate and fire neuron

The idea here is to solve the integrate and fire equation on a computer, the usual way to do that is to divide time into little slices, say δt wide and pretend dV/dt is $\delta V/\delta t$, the change in V divided by the change in t. This gives the so called Euler method:

$$V \to V + \delta V \tag{2}$$

where

$$\delta V = \frac{(E_L - V + R_m I)\delta t}{\tau_m} \tag{3}$$

A Python version

On the Raspberry Pi you can open a text editor by typing `nano`. This is a very simple text editor but it has the advantage that all the commands are listed at the bottom. $\wedge X$, that is control-X, exits and will ask if you want to save your program as you do; there is no other way to save your program. If you want to edit your existing program rather than start a new one you write `nano foo.py` assuming your program is called `foo.py`.

Here is a Python program to do an integrate and fire neuron

```
e_l = -70                                                    1
tau_m = 10                                                   2
v_t = -55                                                    3
                                                             4
r_i = 10                                                     5
                                                             6
delta_t = 0.1                                                7
                                                             8
t=0                                                          9
                                                             10
big_t=2000                                                   11
                                                             12
v=e_l                                                        13
                                                             14
while t<=big_t:                                              15
        v+=(e_l-v+r_i)*delta_t/tau_m                         16
        if v>=v_t:                                           17
                print "|",                                  18
                v=e_l                                        19
        else:                                                20
                print "_",                                  21
                                                             22
        t+=delta_t                                           23
```

The numbers on the right aren't part of the program, it is just handy to have them. Can you work out what the different parts of the program do? To run it, save it, say as `lif.py` and then type `python lif.py`. Try changing the input current. Can you write another program to plot the firing rate, that is the average number of spikes a second, as a function of the input?

To examine the behaviour further we can try plotting the results. One easy way to store the output from a program is called pipping, this isn't as sophisticated as having the program open a file and store the data itself, but it is very straightforward and so it often more convenient. The idea is to send what would normally appear on the screen into a file. In our case we would write `python lif.py > volt.dat` and whatever would have been printed on the screen will not go into `volt.dat`.

What do we want in `volt.dat`, well let's store the voltage, so we replace the while look with

```
                                                             1
while t<=big_t:                                              2
        v+=(e_l-v+r_i)*delta_t/tau_m                         3
        if v>=v_t:                                           4
```

```
            print t,0                                   5
            v=e_l                                       6
    else:                                               7
            print t,v                                   8
                                                        9
    t+=delta_t                                          10
```

To plot the output we can use `gnuplot`; type `gnuplot` to enter gnuplot and the commange `plot 'volt.dat' us 1:2 w lines` plots the second column in the file against the first using a line.