NEURON: the HOC programming language

Bill Lytton

SUNY - Downstate

Brooklyn, NY

Talk to the simulator

- Similar to **C** or **Perl** but *DON'T* use semicolons
- **HOC**=Higher Order Calculator (Kernighan)
- **oc** is an object-oriented augmentation

Numbers

- Integers are handled internally with full precision: 5 same as 5.0
- Can declare an array of numbers: double x[10]
- but vectors are usually better
- Scientific notation uses ’e’ or ’E’
- oc>5e3
  5000
  oc>5E3
  5000

Functions & operators: pluses and minuse

- Functions: sin, cos, tan, sqrt, log, log10, exp
- Arithmetic operators: + - / %
  oc>5+3 // put comment after double slash
- 8
- Logical operators: && || !
- Comparison operators: == != < >
  oc>5==5
- 1
- NB: x=5 vs x==5

NB: x=5 vs x==5

- oc>x = 5 + 7 /* another way to comment */
- oc>x==12
- 1
- oc>x==(5+8)
- 0
- oc>x
- 12

Assignments

- x = x+1
- x += 1
- x *= 2
- NO: x++ (**C** but not in **HOC**)

Block of code

- A section of code that gets executed together
- Can be used in a conditional or a procedure
- Statements surrounded by curly brackets – no separator
- Confusing: { x = 7 print x x = 12 print x }
  7
  12
- Better on individual lines:
  { x = 7
  print x
  x = 12
  print x }

Conditionals and controls

- Decides whether or how often to execute a block
- if (5==5) { print "yes" } else { print "no" }
- did I mention?: ‘if (x=5)’ – you mean ‘if (x==5)’
- while (x<=7) { print x x+=1 }
- for x=1,7 print x
- for (x=1;x<=7;x+=2) print x

proc and func

- proc hello () { print "hello" }
- oc>hello()
  hello
- functions can only return a number
- func hello () { print "hello" return 1 }
- oc>hello()
  hello
  1

Number arguments to procedures:

- proc add () { print $1 + $2 }
- oc>add(5,3)
  8
- func add () { return $1 + $2 }
- print 7*add(5,3)
  56

Strings

- Unlike numbers, string variables must be explicitly declared
- oc>strdef str
  oc>str=5
  nrniv: parse error
  str=5
  oc>str= "hello"
  oc>print str
  hello

Objects

- objref or objectvar declares an object pointer:
  objref g,vec[5],list
- the command *new* creates a new instance of an object
- Graphs, vectors, lists, files are all handled as objects
  g = new Graph()
  for ii=0,4 vec[ii] = new Vector()
  list= new List()
- “dot” notation accesses object components or procedures
  g.erase() // only makes sense if g is a graph
  vec.x[3] // will access a location in vector vec

Simulation commands

- GUI buttons are connected to hoc level commands
- Can create and run simulations form the command line
- oc> create soma
- oc> access soma
- oc> insert hh
- oc> ismembrane("hh")
  1

Sim - stim

- oc> objref stim
- oc> stim = new IClamp(0.5) // current clamp obj
- oc> stim.amp=20 // need big stim (big L, diam)
- oc> stim.dur=1e10 // duration

Sim - running

- oc> tstop = 2 // stop at the peak of the spike
- oc> run()
- oc>print v, v(0.5), soma.v(0.5) // all equivalent
- 38.764279

Vectors

- Can record to vectors and then analyze the contents
- objref vec

```
oc> vec=new Vector()
oc> vec.record(&soma.v(0.5))
oc> tstop = 100
oc> run()
resize_chunk 2046
resize_chunk 4094
resize_chunk 8190
resize_chunk 16382
```

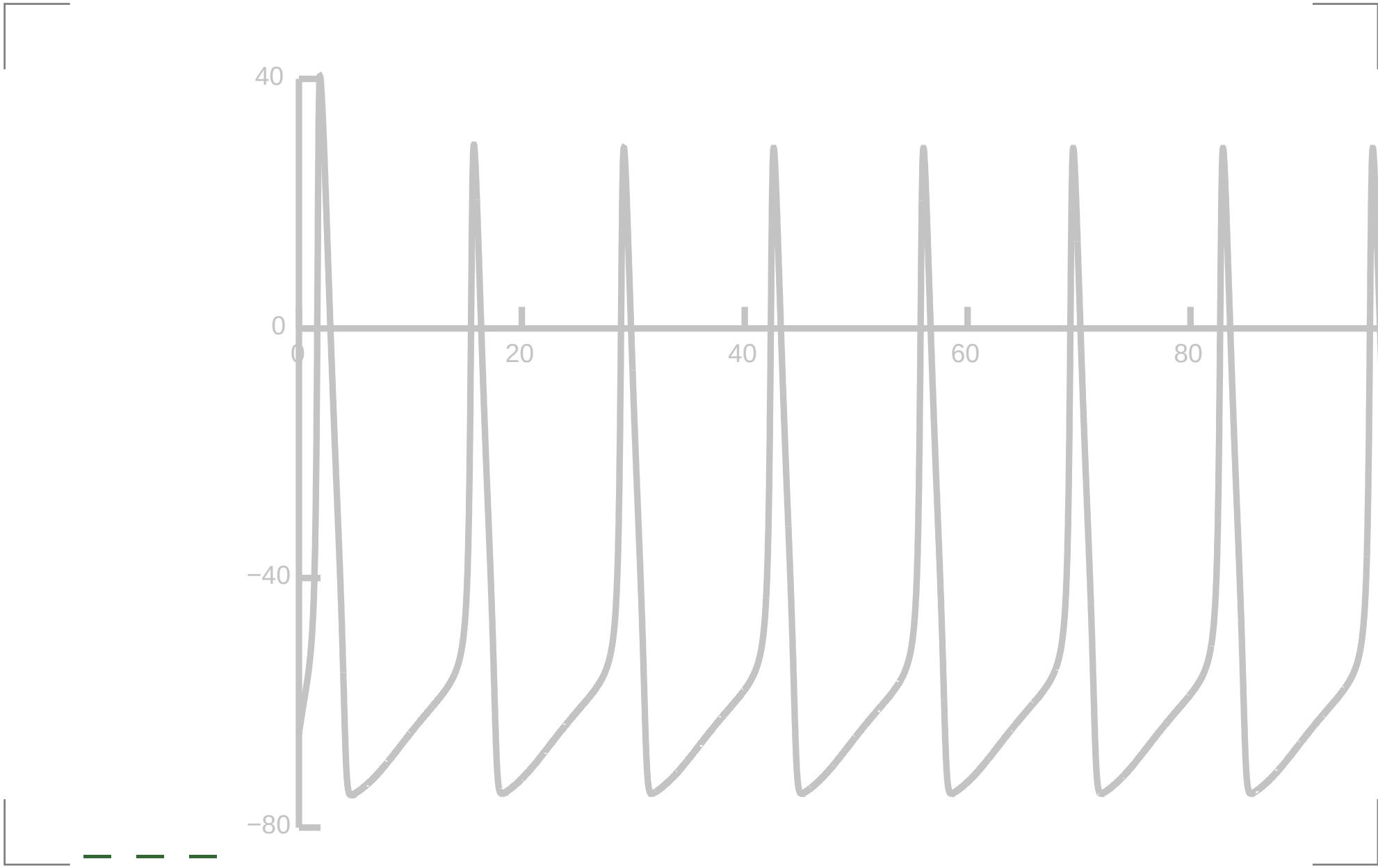
What have we recorded?

- print vec.size(),dt,vec.size*dt,tstop
- print vec.min,vec.max
  -74.774437 40.444033
- print
  vec.min_ind,vec.max_ind,vec.min_ind*dt,vec.max_ind*dt
  470 190 4.7 1.9
- print vec.x[470],vec.x[190]
  -74.774437 40.444033

Can analyze signals using vectors

- Find the steepest action potential
- vec[1].deriv(vec,dt)
- print vec[1].max_ind,vec[1].max_ind*dt
  168 1.68

Quick & dirty graphics



Graphing a vector

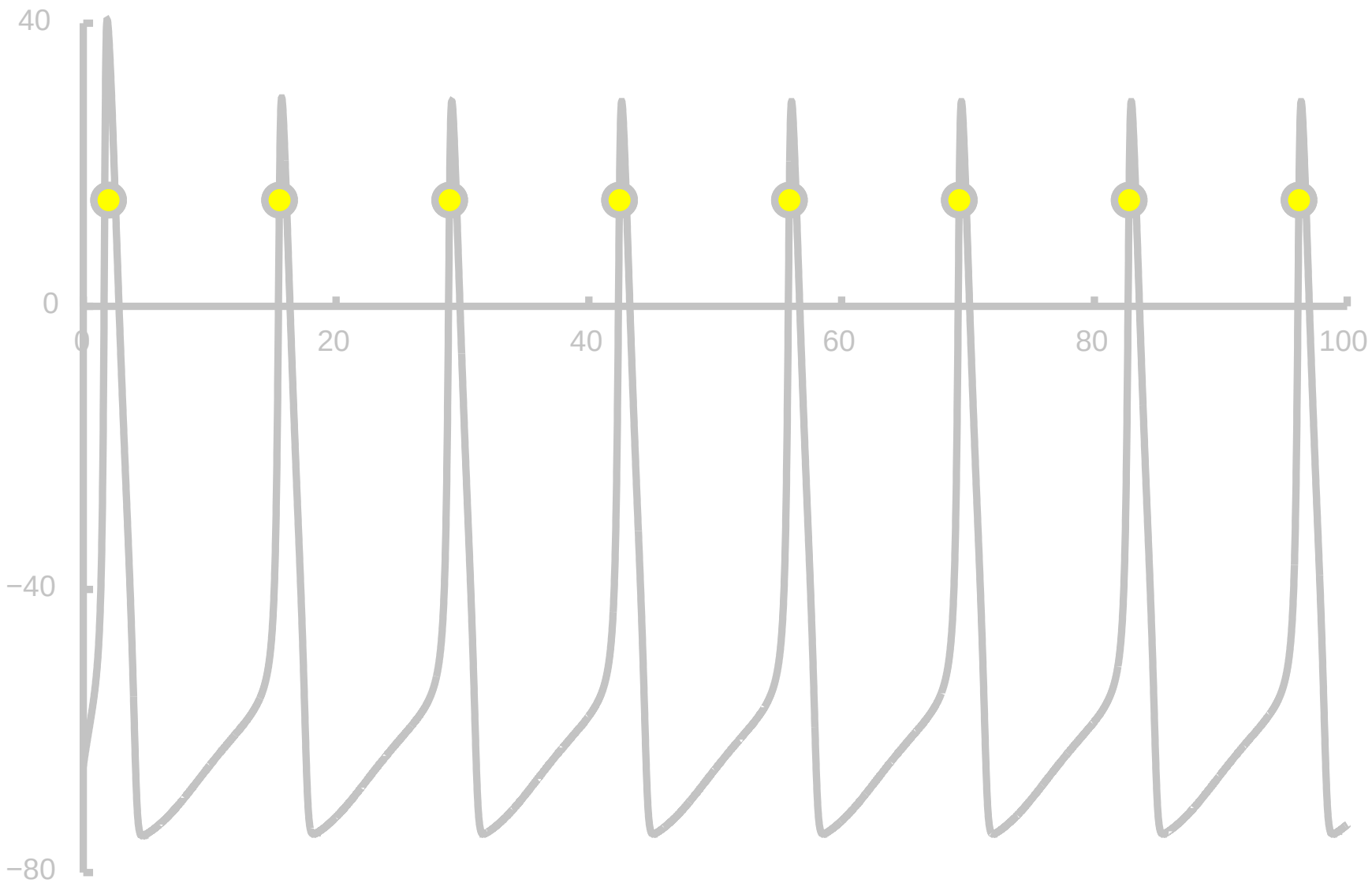
- Can put up a graph from the main menu or by hand
  g = new Graph()
- Draw the vector on the graph
  vec.line(g,dt)
- Need a time vector if using var dt
- Erase and redraw
  g.erase

Find spikes

- vec[1].indvwhere(vec,">",15) // indices above a threshold
- vec[1].mul(dt) // times
- spktime=0
- for ii=0,vec[1].size-1 if (vec[1].x[ii]<spktime+2) vec[1].x[ii]=-1 else spktime=vec[1].x[ii]
- vec[2].where(vec[1],">",0)

Check results graphically

```
for     ii=0,ind.size-1     g.mark(vec[2].x[ii],15,"O")
```



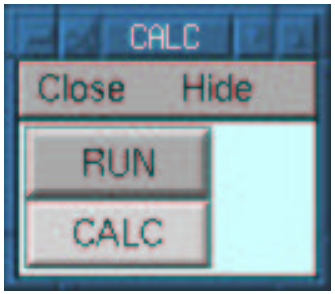
Now can calculate means etc.

- calculate differences: vec[3].sub(othervec)
- take inverses: vec[3].resize(), vec[3].fill(1), vec[3].div(othervec)
- print vec[3].mean(), vec[3].stdev()

Other useful vector functions

- vec.setrand(rdm) // where rdm=new Random()
- vec.fft() // fast fourier transform
- vec.sort()
- vec.histogram()
- vec.apply("user_func")

Putting up buttons



Reading and writing files

- file=new File()
- file.wopen("tmp")
- vec.printf(file) // or vec.vwrite(file) for binary
- file.close()

TOC