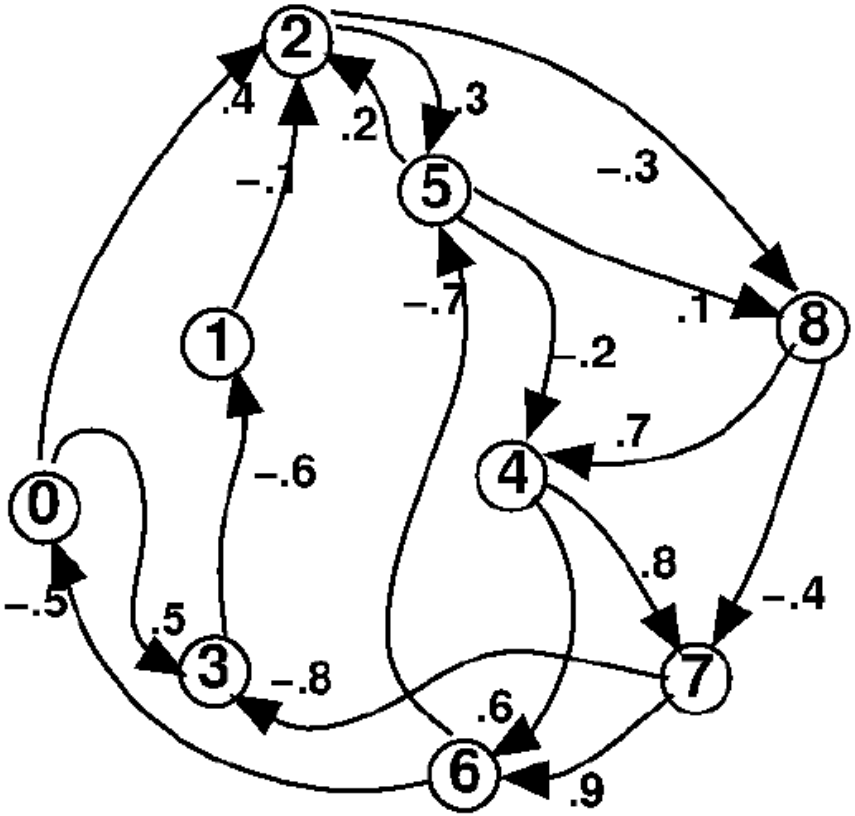
Wiring networks in Neuron

Bill Lytton

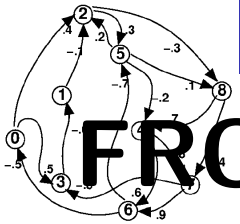
Explore (poke) the computer

- This is not chem lab
- Try to make lots of mistakes
- Keep a notebook of your mistakes (and successes)
- You will get frustrated – just try something else

Simple network

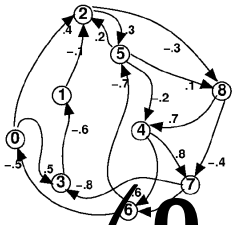


connections table



FROM ⇒ / TO ⇓	A	B	C	D	E	F	G	H	I
A	▬						-.5		
B		▬		-.5					
C	.4	-.1	▬			.2			
D	.5			▬				.2	
E					▬	.2			.7
F			.3			▬	.2		
G					.1		▬	.9	
H					.7			▬	.2
I			-.3			.1			▬

connectivity matrix



$$\begin{pmatrix} \mathbf{0} & 0 & 0 & 0 & 0 & 0 & -.5 & 0 & 0 \\ 0 & \mathbf{0} & 0 & -.5 & 0 & 0 & 0 & 0 & 0 \\ .4 & -.1 & \mathbf{0} & 0 & 0 & .2 & 0 & 0 & 0 \\ .5 & 0 & 0 & \mathbf{0} & 0 & 0 & 0 & .2 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{0} & .2 & 0 & 0 & .7 \\ 0 & 0 & .3 & 0 & 0 & \mathbf{0} & .2 & 0 & 0 \\ 0 & 0 & 0 & 0 & .1 & 0 & \mathbf{0} & .9 & 0 \\ 0 & 0 & 0 & 0 & .7 & 0 & 0 & \mathbf{0} & .2 \\ 0 & 0 & -.3 & 0 & 0 & .1 & 0 & 0 & \mathbf{0} \end{pmatrix} \begin{pmatrix} A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \\ I \end{pmatrix}$$

Define connectivity

- S=number of syns; D=divergence; C=convergence
- $S = C \cdot Post$; $S = D \cdot Pre$
- connection density
  $p_{ij} = \frac{S}{(Pre \cdot Post)} = \frac{C}{Pre} = \frac{D}{Post}$
- Below: 1 kind of cell $\Rightarrow$ Pre and Post are the same

Visual vs textual approachs

- scalability
- verifiability
- portability

Where to start?

networks are daunting

- NetPyNE
- Prototype network with Artificial Cells
- Download network from modelDB and rewrite
- Write network code in favorite language and generate connectivity matrix

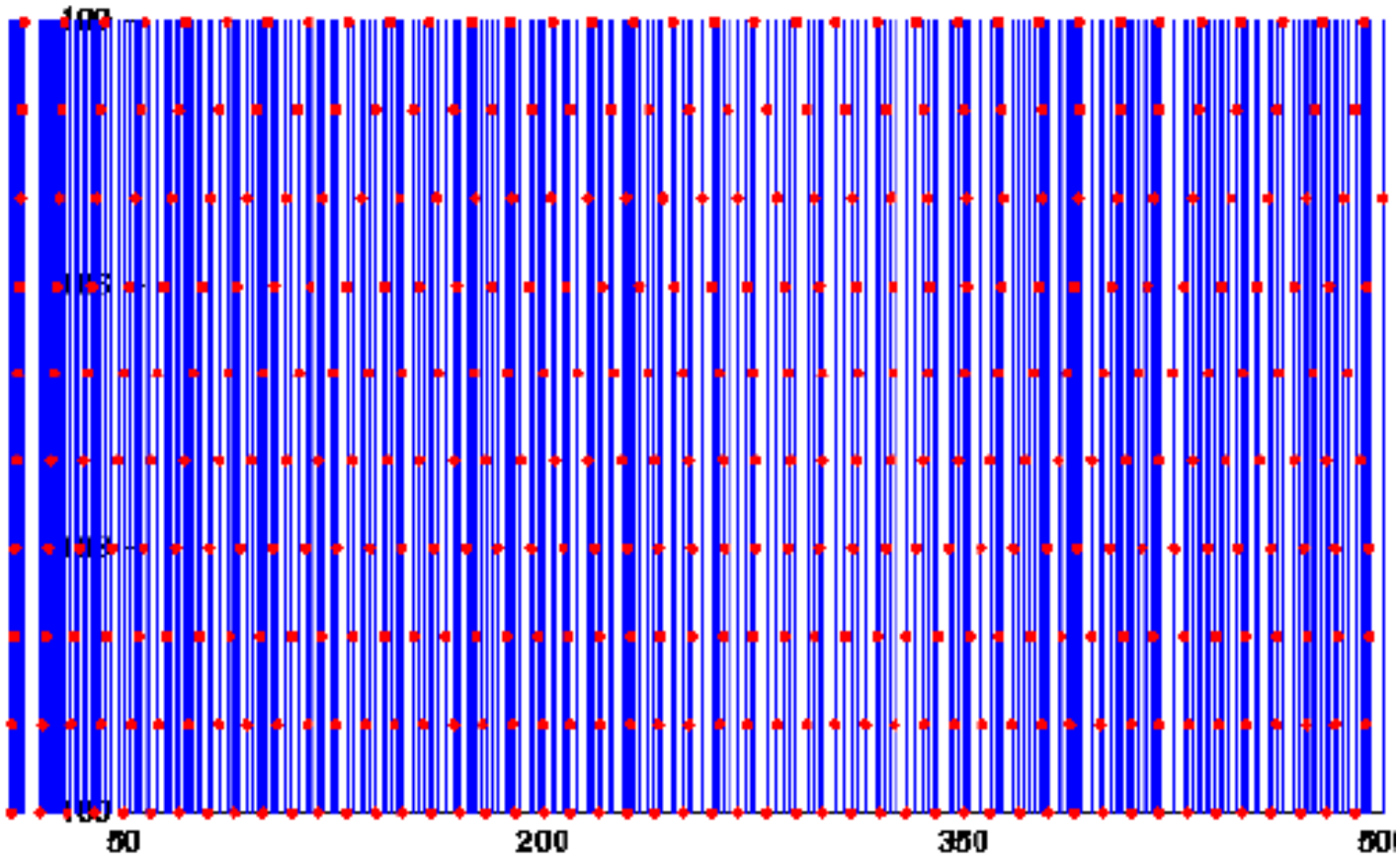
Hopfield-Brody synchronization model

- All to all connectivity
- Firing cells synchronize due to mutual inhibition
- Each cell has a natural period
- Inhibition from other cells provides a reset, locking them together

Unconnected cells

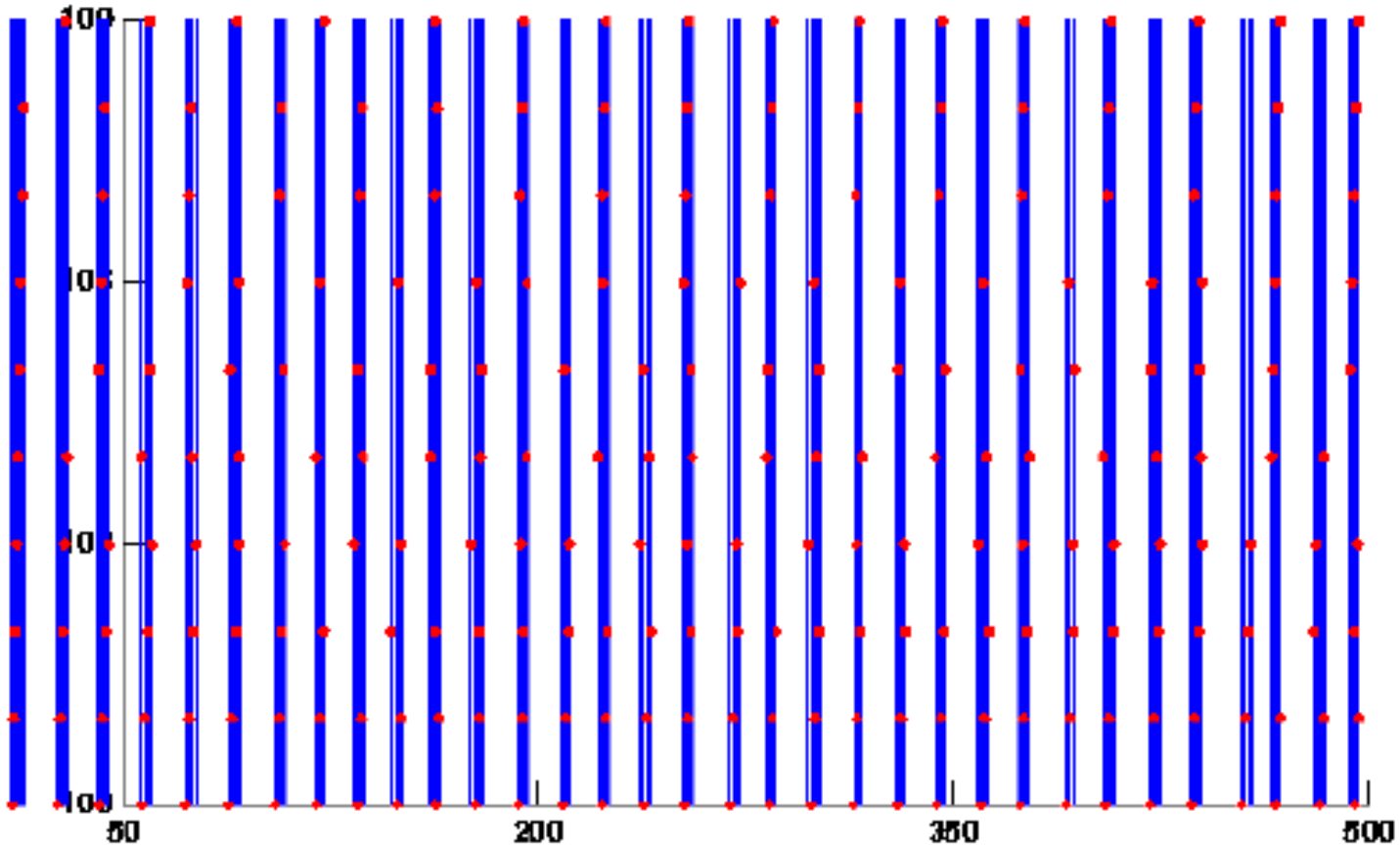
wt = 0 # weight

cell fire at own rates and go out of phase



Negative (inhibitory) connectivity

wt=-3e-1



Let's look at the code

```
class Cell:
    def __init__ (self, _id, _x, _y, _z):
        self.pp = h.INVLF()
        self.pp.tau, self.pp.invl = 10, 20
        self.id, self.x, self.y, self.z = _id, _x, _y, _z

def createnet (ncell=100):
    global ncells, cells, nclist
    ncells = ncell
    cells = [Cell(i,i,0,0) for i in range(ncell)]
    nclist = [h.NetCon(x[0].pp,x[1].pp) for x in permut(cells,2)]
    for ce in cells:
        nc=h.NetCon(ce.pp, None)
        nc.record(tvec, ind, ce.id)
        nclist.append(nc)

def setparams (w=None, delay=None, tau=None, low=None, high=None):
    """params for both netcons and cells"""
    for n in nclist:
        if w: n.weight[0]=w
        if delay: n.delay=delay
    for ce,ran in zip(cells, np.random.uniform(low if low else 10,
        ce.pp.invl = ran
        if tau: ce.tau = tau
```

Q&D synchronization measure

many ways to measure sync

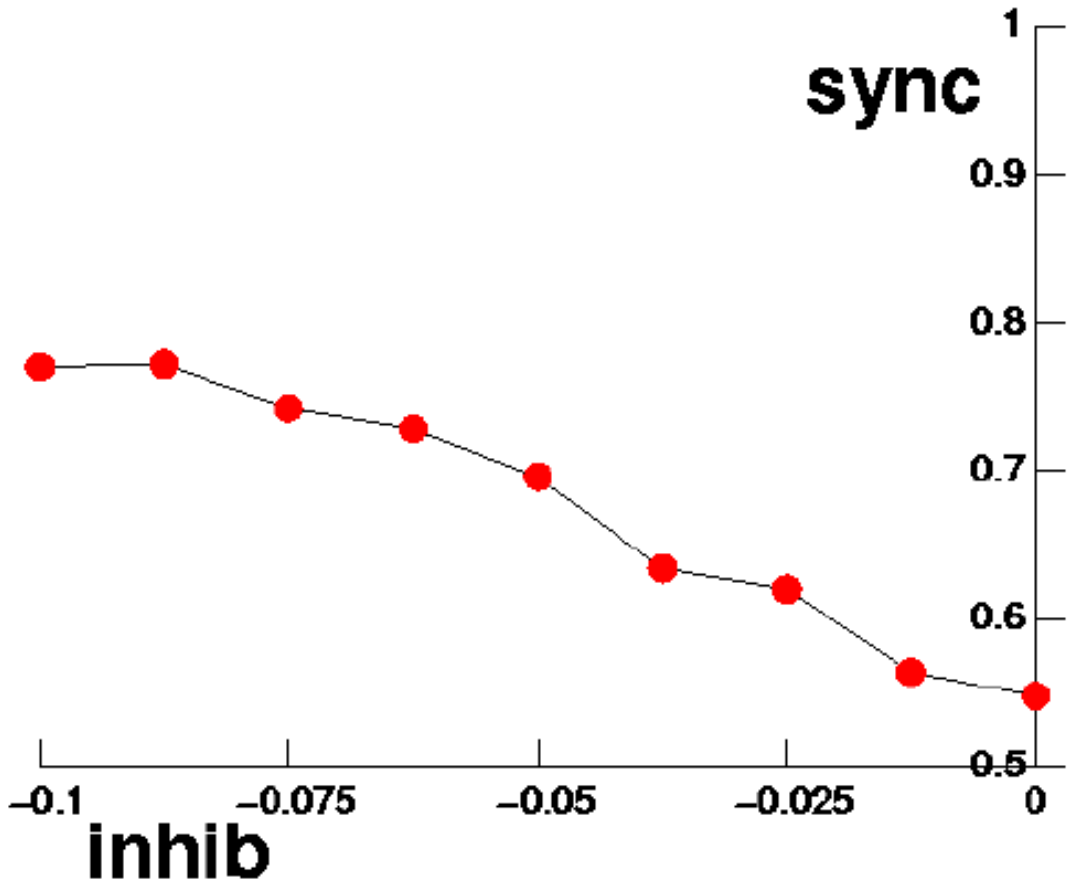
```
def syncer (tvec=h.tvec ,width=1):
    t0=-1; cnt=0
    for tt in tvec:
        if tt>=t0+width: t0=tt; cnt+=1
    return 1.0-cnt/(h.tstop/width)
```

Check sync with increasing inhib

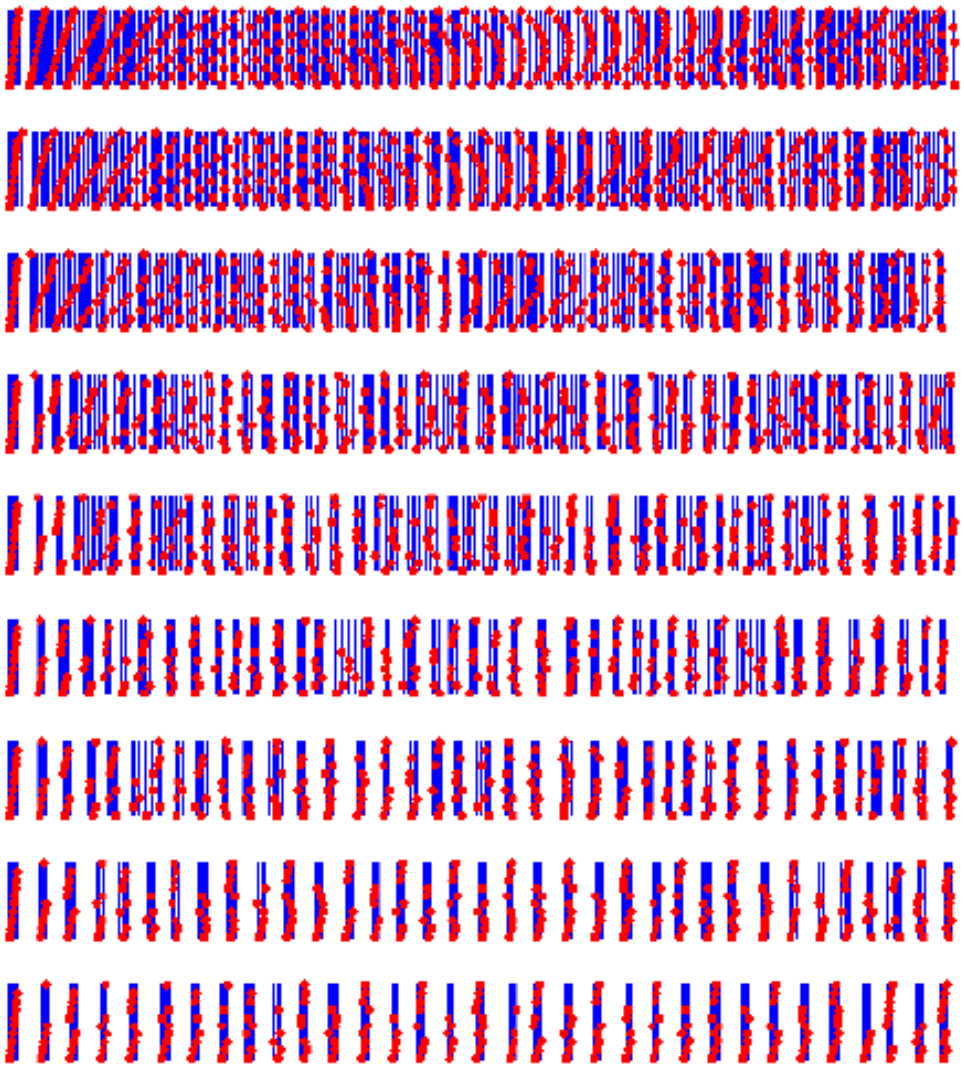
```
max= -0.1
for w in numpy.linspace(max,0.0,8):
    setparams(); h.run()
    vec[1].append(w)
    vec[0].append(syncer())
```

Graph results

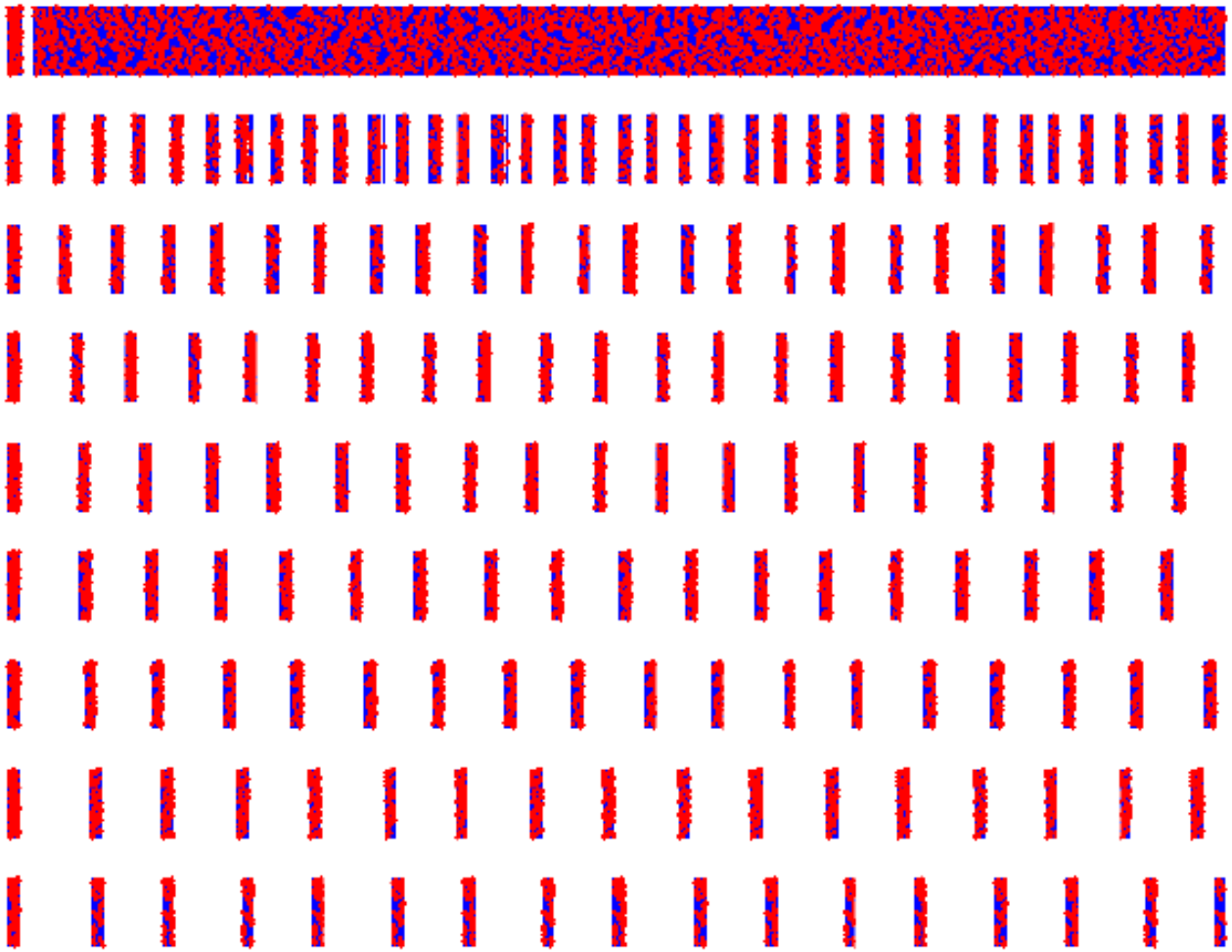
```
{vec.line(g,vec[1]) vec.mark(g,vec[1])}
```



Confirm with raw data

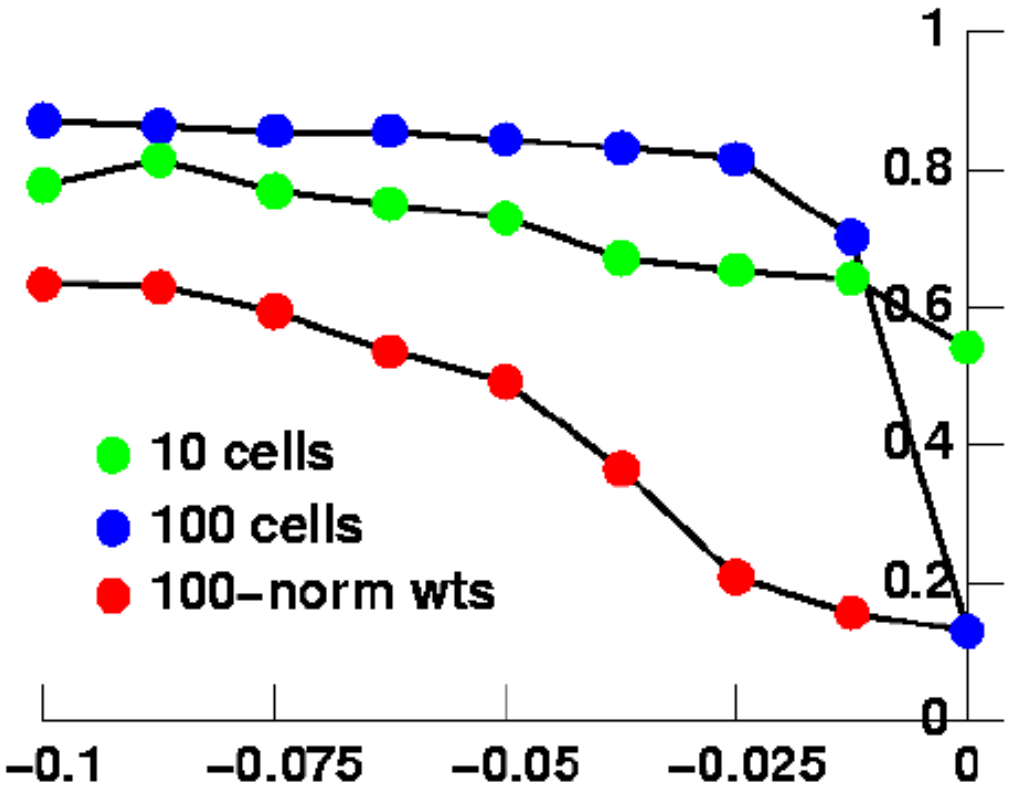


Scale up to 100 cells



Need to normalize weights

w=ii/ncell



Often make use of index vectors

- Use codes for ntype different cell types: PYR=0;OLM=1;BAS=2
- Have certain number of cells in each group: numc[BAS]=10
- Create a list of Vectors: vec=[h.Vector() for i in range(ntype)]
- Use the vectors as indices into the cell types: vec[BAS].indgen(0,numc[BAS])

Wiring the network

```
# wire() full connectivity
def wire ():
    nclist.remove_all()
    for pre in cells:
        for post in cells:
            if pre!=post:
                nclist.append(
                    h.NetCon(pre.pp,post.pp))
```

Rewiring for different densities

- Can rewire entire network – slow
- rewrite wire():

```
# don't create if syn_num is not on list
for i,pre in enumerate(cells):
  for j,post in enumerate(cells):
    if i!=j and
    not vec[0].contains(i*100+j):
      ... new NetCon ...
```

Or rewire on the fly by setting weights to 0

- Index synapse from 0 $\rightarrow S =$ ncells$^2 -$ ncells
- Zero out $(1 - p_{ij}) \cdot S$ syns (30% connectivity, zero out 70%)
- *e.g.*,

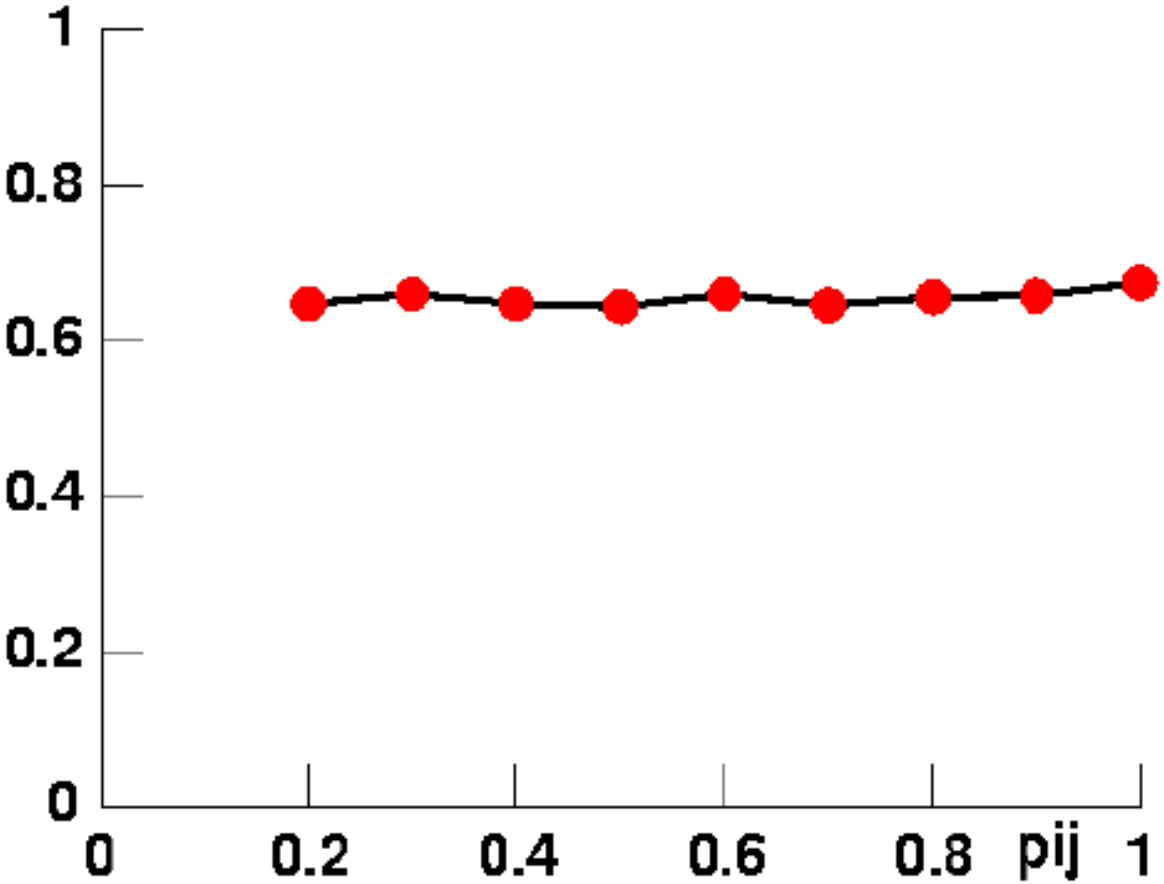
```
rdm[0].discunif(0,S-1) //
random indices
vec[0].resize((1-pij)*S)
vec[0].setrand(rdm[0])
```

Rewrite weight()

Reset weights rather than rewiring

```
# weight2(WT, EXCLUDE_VEC)
#  0 unless in EXCLUDE_VEC
def weight2 (w, exclude):
    for nc in nclist {
        if exclude.contains(i):
            nc.weight[0]=0
        else:
            nc.weight[0]=w
```

Density doesn't matter?!



Checking connectivity

- Surprising findings are often artifacts
- Pseudo-random may be more pseudo than random
- Best-written programs of mice & men
- Check if network looks reasonable

cvode.netconlist()

To access NEURON's internal NetCon list

- can have 3 or more levels of representation – debug by looking at the base
- Unwieldy, but useful to verify internally
- To check divergence: for c in cells: print h.cvode.netconlist(c.pp,"","").count()
- To check convergence: for c in cells: print cvode.netconlist("","",c.pp).count()
- Could count non-zero synapses by iterating through cvode.netconlist("","","")

Working with parallel databases

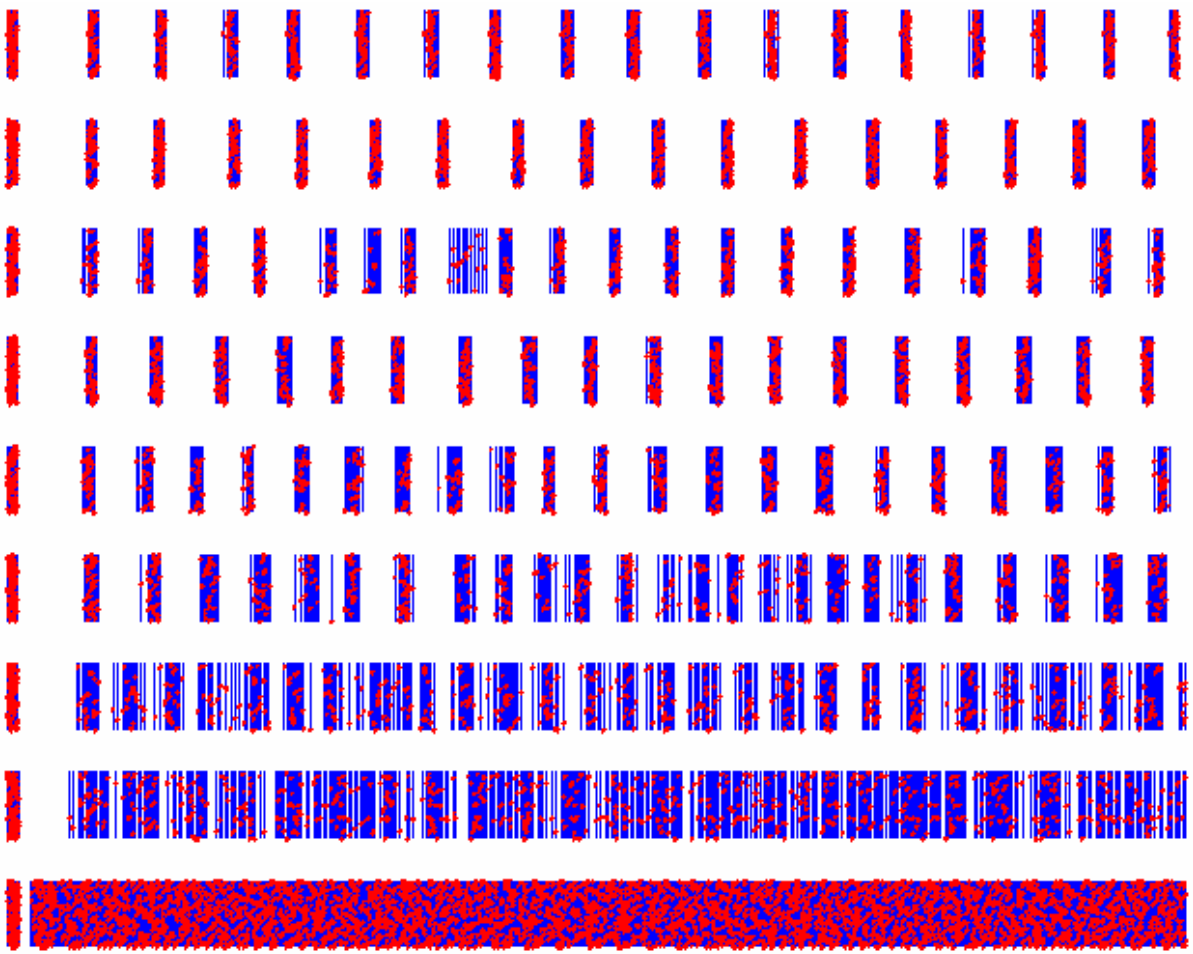
- now have yet another level of representation
- Might have or build a connmat
- Compare back to cvode.netconlist()
- Beware stray NetCons

I checked non-zero connectivity

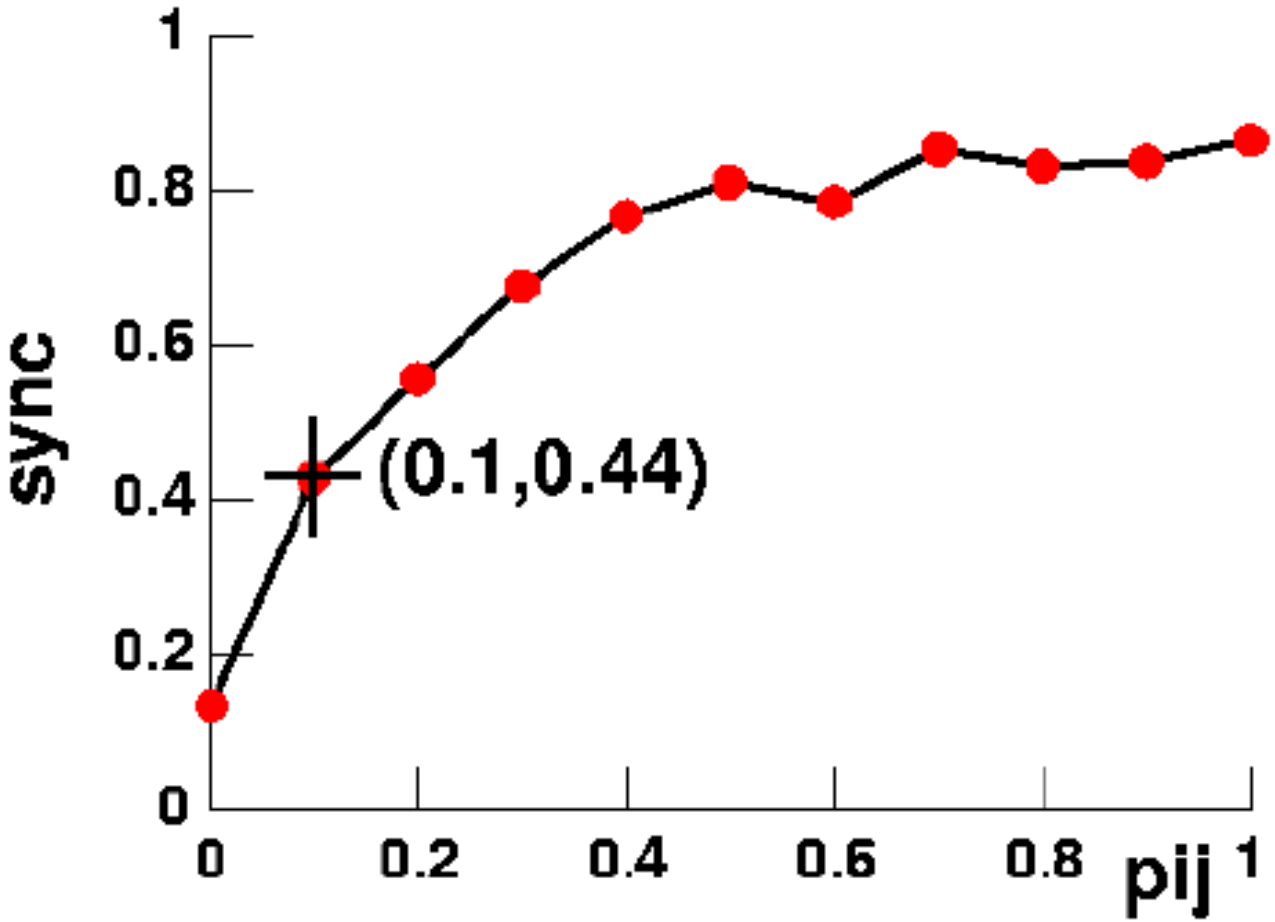
- $p_{ij} = 0.2 \ldots$
- fconn(vec[4],vec[5])
- print vec[4].size

  4479
- Wrong answer!: $p_{ij} \cdot S \sim 2000$
- Zero-weighting vector had repeats

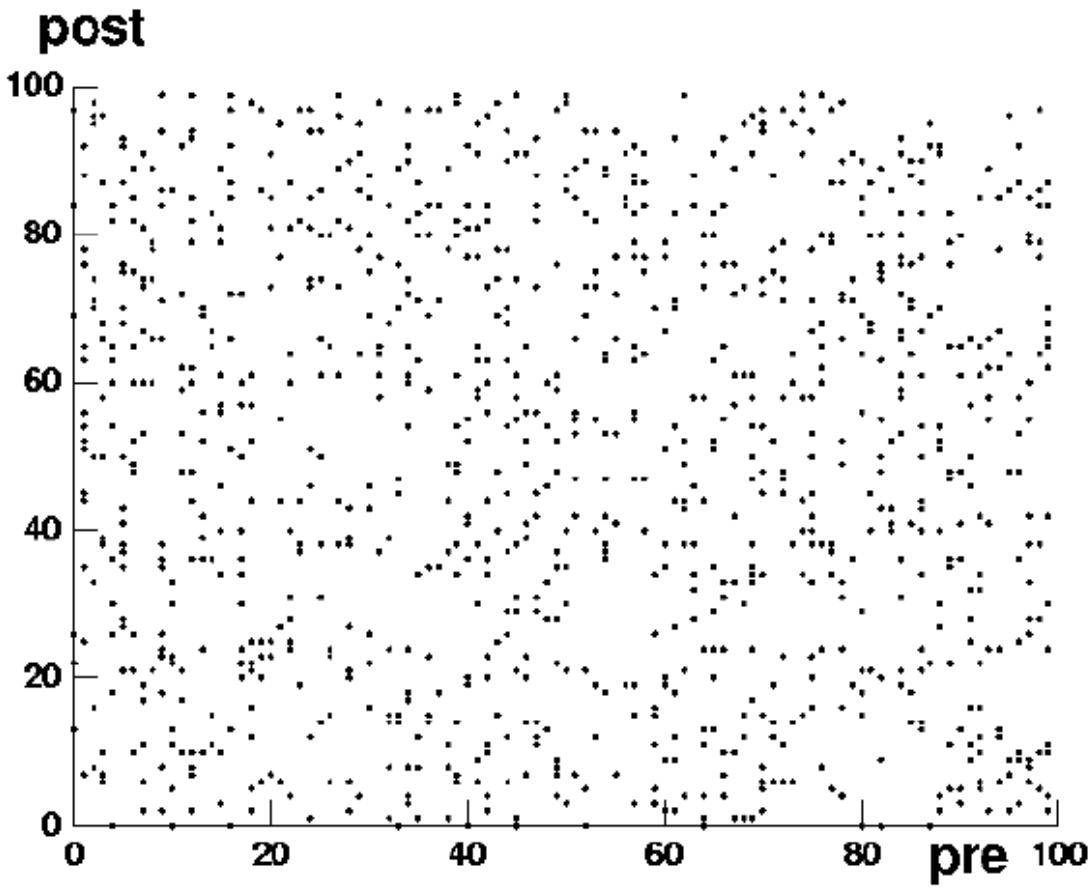
Problem was the randomizer – wasn't finding unique values



Synchronization measure



Look at 10% connectivity



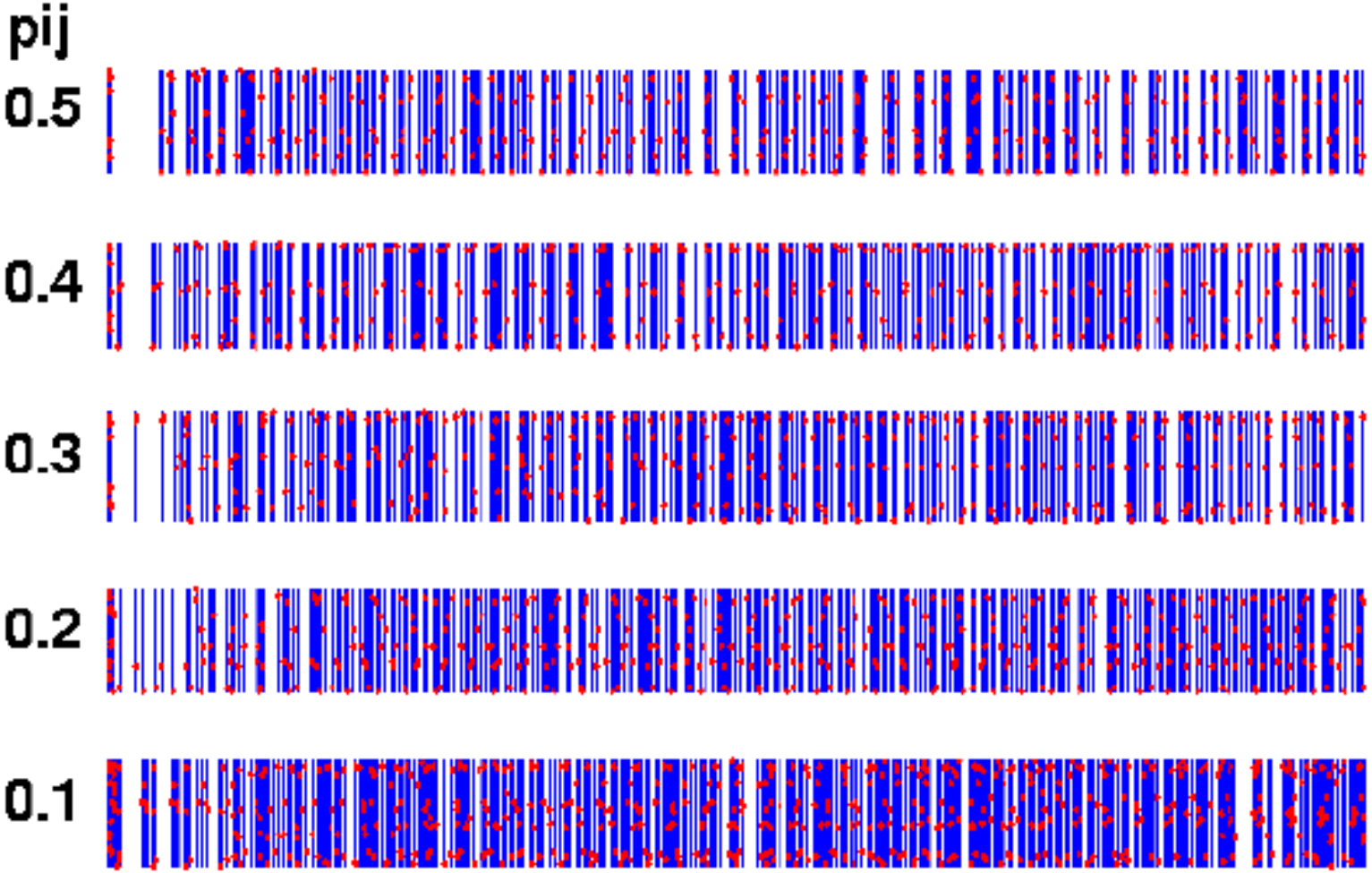
Balancing convergence and divergence

- Wide variation in connectivity:
  $< C >$=9.91±2.99; $< D >$=9.91±3.09
- C_{min}=3; C_{max}=21; D_{min}=4; D_{max}=17;
- With realistic cells, unbalanced convergence will blast some cells and leave others out

Geographic connectivity

- Neuroanatomy furnishes non-random connectivity
- Map onto model connectivity
- *e.g.*, fall-off with distance

Sparse connectivity with distance falloff doesn't sync well



Other explorations

- Try weight fall-off rather than restricting wiring
- Mix excitatory and inhibitory connects
- Connect two populations at various densities