Introduction to Neural Networks
U. Minn. Psy 5038

Lateral inhibition

Introduction

Last time

- **Developed a "structure-less, continuous signal, and discrete time" generic neuron model and from there built a network.**
- **Basic linear algebra review. Motivated linear algebra concepts from neural networks.**

Today

We are going to look at an explanation of a human perceptual phenomenon called Mach bands, that involves a linear approximation based on a real neural network. This is an example of neural filtering found in early visual coding. We will study two types of network that may account for Mach bands: 1) feedforward; 2) feedback. The feedback system will provide our first example of a dynamical system. The system we will look at was developed as a model of the neural processing in the horseshoe crab (*limulus*) compound eye. Despite the (apparent) enormous difference between your visual system and that of the horseshoe crab, our visual system shares a fundamental image processing function with that of this lowly crustacean (and virtually all other animals that process image patterns).

- **An application of a simple linear model for visual spatial filtering**
- **Add some dynamics for visual spatial filtering**
- **Winner-take-all network: Add a threshold non-linearity**

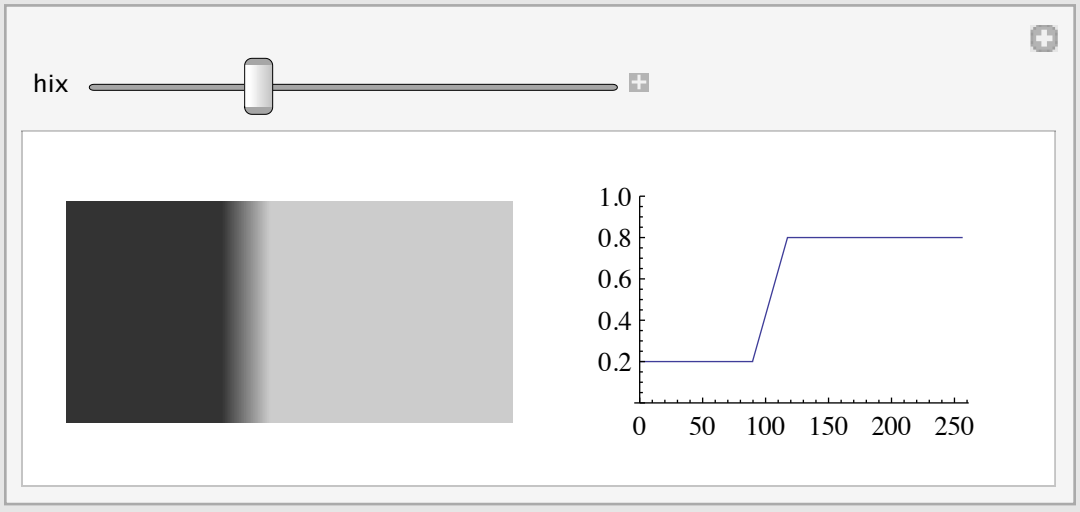
Mach bands & perception

Ernst Mach was an Austrian physicist and philosopher. In addition to being well-known today for a unit of speed, and *Mach's Principle* in theoretical physics, he is also known for several visual illusions. One illusion is called "*Mach bands*". Let's make and experience some.

In[13]:=
```
width = 256;
y[x_, hix_] := Module[{low, hi, lowx},
    low = 0.2; hi = 0.8; lowx = .35 * width;
    Piecewise[{{low , x < lowx},
      {((hi - low) / (hix - lowx)) x - ((hi - low) lowx) / (hix - lowx) + low ,
       x >= lowx && x < hix}, {hi, x >= hix}}]];
```

In[16]:=
```
Manipulate[
 e3 = Table[y[i, hix], {i, 1, width}]; picture2 = Table[e3, {i, 1, 60}];
 GraphicsGrid[
  {{Graphics[Raster[picture2, {{1, 1}, {120, 60}}, {0, 1}]],
    Plot[y[x, hix], {x, 1, width}, PlotRange → {0, 1}]}}],
 {{hix, 87}, width / 3., width * 3 / 4}]
```

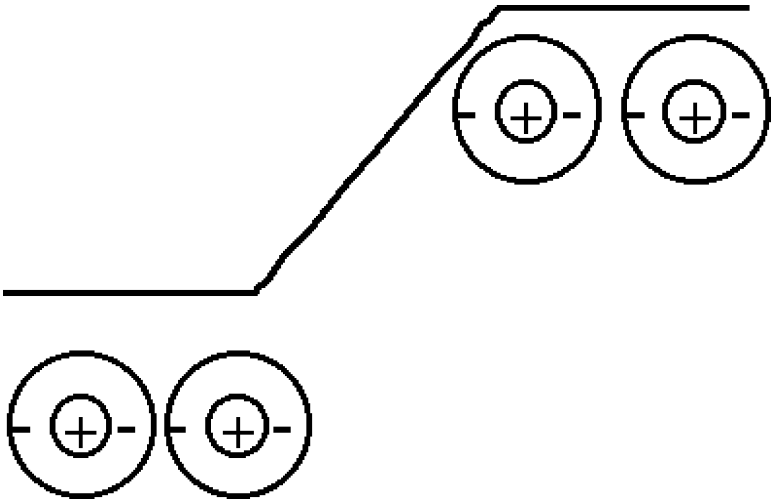
Out[16]=



PlotRange is used to scale the brightness.

What Mach noticed was that the left knee of the ramp looked too dark, and the right knee looked too bright to be explained by the light intensity. Objective light intensity did not predict apparent brightness.

■ **Mach's explanation in terms of lateral inhibition**



In general, lateral inhibition increases contrast at edges.

Neural basis?

Early visual neurons (e.g. ommatidia in horseshoe crab, ganglion cells in the mammalian retina and even later cells in the lateral geniculate neurons of the thalamus, and some cells in V1 or primary visual cortex of the monkey) have receptive fields with Mach's center surround organization. I.e. approximately circular excitatory centers and inhibitory surrounds. Or the opposite polarity, inhibitory centers and excitatory surrounds.

Some history:

Limulus (horseshoe crab)--Hartline won the 1967 Nobel prize for this work that began in the 20's with publications up to the 70's.

(See http : // www.mbl.edu/animals/Limulus/vision/index.html).

Frog -- Barlow, H. B. (1953). Summation and inhibition in the frog's retina. J Physiol, 119, 69-88.

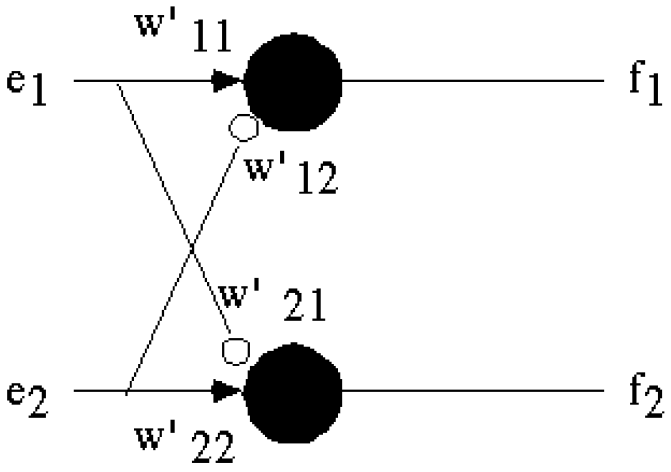
Cat --S. W. Kuffler (1953). Discharge patterns and functional organization of mammalian retina . Journal of Neurophysiology, 16:37--68.

Feedforward model

Two types of models: feedforward and feedback (in our context, "recurrent lateral inhibition")

f = *w′*.e

where **e** is a vector representing the input intensities (the **e1** or **y[]** values above), ***w′*** is a suitably chosen set of weights (i.e. excitatory center and inhibitory surround as shown in the above figure), and **f** is the output.

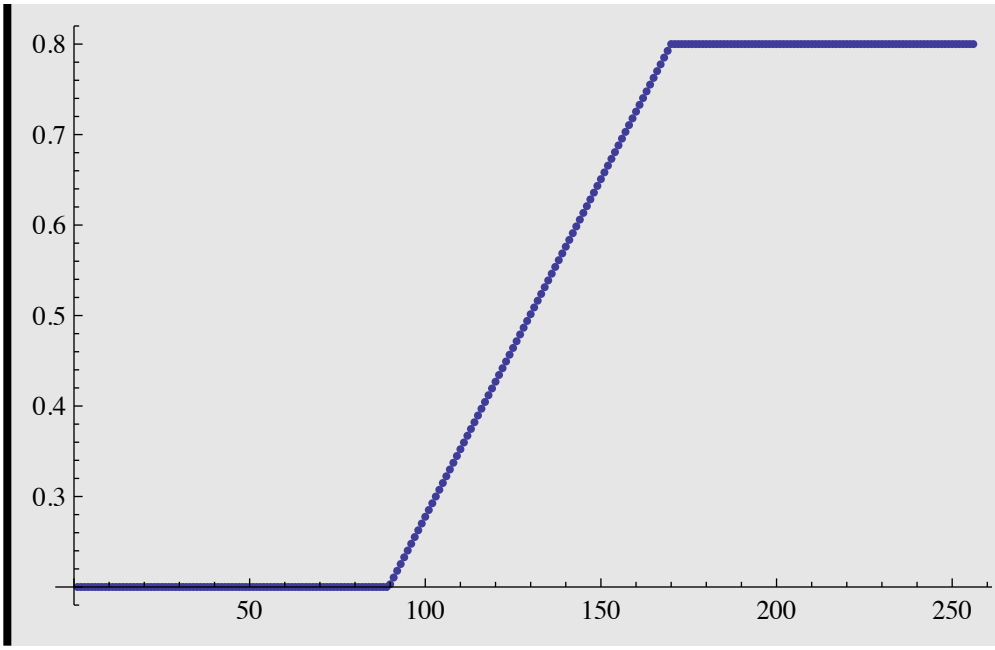


■ Programming implementation

Because changes in the stimulus are one-dimensional, we'll simulate the response in one dimension. We specify the input vector **e** as above:

In[17]:=
```
e := Table[y[i, 170], {i, 1, width}];
ListPlot[e]
```

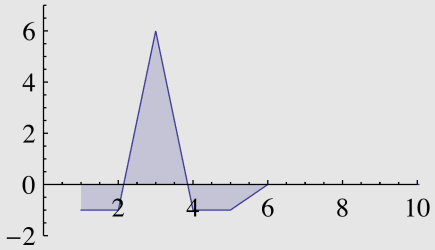
Out[18]=



Let's assign weights consistent with Ernst Mach's 19th century hypothesis. We can model the receptive field for one output unit to be represented by 5 weights, with a center value of 6, and surround values of -1:

In[19]:=
```
wp = Table[0, {i, 1, Length[e]}]; wp[[1]] = -1; wp[[2]] = -1; wp[[3]] = 6;
wp[[4]] = -1; wp[[5]] = -1;
ListPlot[wp, Joined → True, PlotRange → {{0, 10}, {-2, 7}},
 Filling → Axis]
```

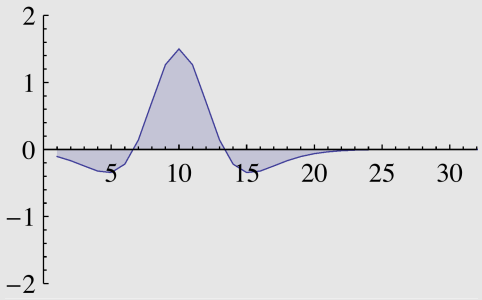
Out[19]=



...or we could get a little more sophisticated, and use a common formula to model center-surround organization, the difference of two Gaussian functions, with different widths. Here is a specific choice that specifies a wider filter than above:

In[20]:=
```
wp = Table[2.5 * Exp[-((i - 10) / 3)^2] - Exp[-((i - 10) / 6)^2],
    {i, 1, Length[e]}];
ListPlot[wp, Joined → True, PlotRange → {{0, width / 8}, {-2, 2}},
 Filling → Axis]
```

Out[21]=



The plot shows the "center-surround" organization of the filter. It is sometimes refered to as a "Mexican hat" filter.

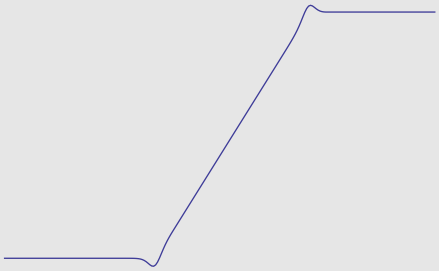
Now assume that all units have the same weights, and calculate the response at each point by shifting the weight filter **wp** right one by one, and taking the dot product with the input pattern **e**, each time:

In[22]:=
```
response = Table[RotateRight[wp,i].e,{i,1,width-30}];
```

This way we can mimic the response we want:

In[23]:=
```
ListPlot[response, Joined → True, Axes → False]
```

Out[23]=



Note that we cut the rotation short to eliminate boundary effects.

Show that you can do this operation as matrix multiplication, where each subsequent row of a matrix W is the vector wp shifted over by one.

■ Convolution

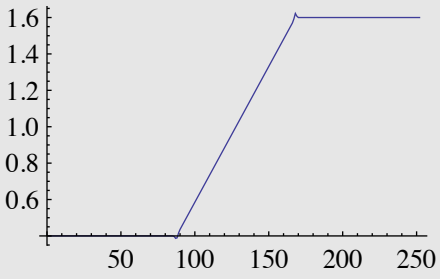
This kind of operation where the same filter gets applied repeatedly at different positions is common in signal processing. *Mathematica* has a function **ListConvolve[ ]** that does this for you. It has additional arguments that allow for handling of the boundaries. What should you do when the filter gets close to the end of the stimulus? A common default is to let the filter wrap around. Another common solution is to "pad" the ends of e with fixed values, such as zero.

What does the retina do?.

Here's ListPlot[] with the simpler center-surround receptive field, {-1, -1, 6, -1, -1}. In mathematics, this filter vector is sometimes called the "kernel" of the filter.

In[24]:=
```
ListPlot[ListConvolve[{-1, -1, 6, -1, -1}, e], Joined → True]
```

Out[24]=



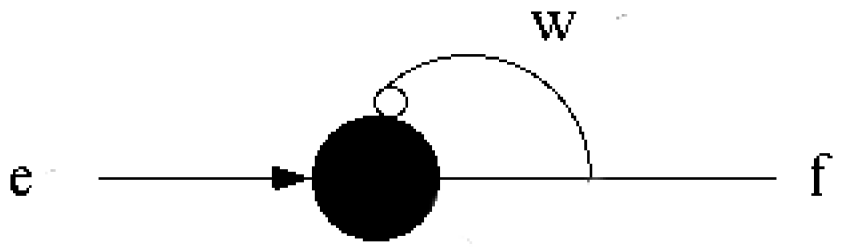
What is the response to the ramp if the sum of the weights is zero? Build a simple edge detector. Let kern={-1,2,-1} and use ListConvolve[ ].

Feedback model: Recurrent lateral inhibition

Now we'll develop a different, dynamical model for lateral inhibition that includes time and feedback. There is neurophysiological evidence for an implementation of lateral inhibition via feedback, called *recurrent lateral inhibition*.

■ Dynamical systems: difference equation for one neuron

State of neuron output **f** at discrete time k.



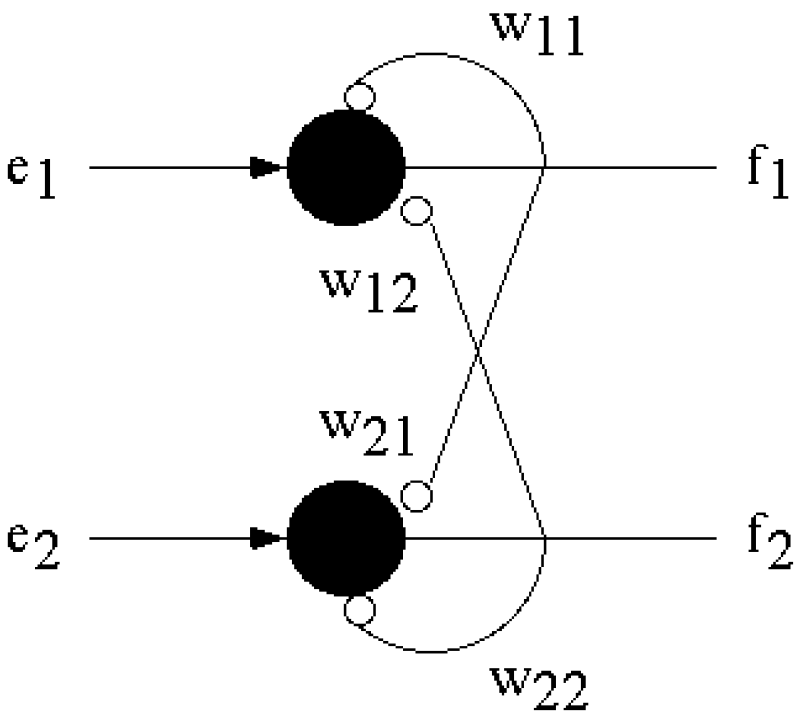
One neuron

$$f[k+1] = e[k] + w\, f[k] \qquad (1)$$

Suppose the initial state f[0] is known and e[k] is zero, can you find an expression for f[k]? What happens if w is less than one? Greater than one?

■ Dynamical systems: Coupled difference equations for interconnected neurons

Consider a two neuron system. The formalism will extend naturally to higher dimensions. To keep this simpler, the weights for the inputs **e** are fixed at one, but we will specify weights for the newly added feedback connections:



Let **e** be the input activity vector to the neurons, **f** is the n-dimensional state vector representing output activity and **W** is a fixed nxn weight matrix. Then for a two neuron network we have:

$$\begin{aligned} f_1[k+1] &= e_1[k] + w_{12}\, f_2[k] + w_{11}\, f_1[k] \\ f_2[k+1] &= e_2[k] + w_{21}\, f_1[k] + w_{22}\, f_2[k] \end{aligned} \tag{2}$$

or in terms of vectors and matrices

$$\begin{pmatrix} f_1[k+1] \\ f_2[k+1] \end{pmatrix} = \begin{pmatrix} e_1[k] \\ e_2[k] \end{pmatrix} + \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} \begin{pmatrix} f_1[k] \\ f_2[k] \end{pmatrix} \tag{3}$$

or in summation notation:

$$f_i[k+1] = e_i[k] + \sum_j w_{ij} . f_j[k] \tag{4}$$

or in concise vector-matrix (and *Mathematica*) notation:

$$\mathbf{f}[k+1] = \mathbf{e}[k] + \mathbf{W} . \mathbf{f}[k] \tag{5}$$

where $\mathbf{W} = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix}$

This equation is an example of a simple dynamical system, with state vector **f**. As you might imagine, the state of a dynamical system typically changes with time (i.e. iteration k).

Are there solutions for which the state does not change with time? If there are, these solutions are called *steady state* solutions.

In contrast to the way we set up the weights for the feedforward matrix (which included the forward excitatory weights), we are going to assume later that all of these weights are inhibitory (because we are modeling lateral *inhibition*). The positive contributions, if any, will come from the input **e**.

Steady state solution for a discrete system is mathematically equivalent to a linear feedforward model

A steady-state solution simply means that the state vector **f** doesn't change with time:

$$\mathbf{f}[k+1] = \mathbf{f}[k] \tag{6}$$

or in vector and *Mathematica* notation:

$$\mathbf{f = e + W.f}$$

where we drop the index k. Note that by expressing **f** in terms of **e**, this is equivalent to another linear matrix equation, the feedforward solution:

$$\mathbf{f = W'.e},$$

where

$$\mathbf{W' = (I - W)^{-1}}$$

The **-1** exponent means the inverse of the matrix in brackets. **I** is the identity matrix, which in two dimensions is: $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

We will review more later on how to manipulate matrices, find the inverse of a matrix, etc.. But for the time being, think of an identity matrix as the generalization of unity: **1**. **1** times **x** returns **x**.

The point of the above derivation is that we can obtain a steady-state solution for recurrent inhibition (i.e. with feedback) which is equivalent to non-recurrent linear network (no feedback, just feedforward). In general, there may be more than one underlying neural circuit to explain the external behavior of a network. Telling them apart requires doing the right kind of experiment.

■ Dynamical system -- coupled differential equations ("limulus" equations)

In our discussion of the different types of neural models, we noted that continuous time is a more realistic assumption for a neural network.

So what if time is not modeled in discrete clocked chunks? It is straightforward to extend the discrete time model to a continuous time model. But then to simulate the dynamics,

we'll go back to a discrete approximation, but keep in mind that the behavior might depend on the temporal grain of our approximation.

The theory for coupled discrete equations

$$\mathbf{f}[k+1] = \mathbf{e}[k] + \mathbf{W}.\mathbf{f}[k] \tag{7}$$

parallels the theory for continuous differential equations where time varies continuously:

$$\frac{d\mathbf{f}}{dt} = \mathbf{e}[t] + \mathbf{W''}.\mathbf{f}[t] \tag{8}$$

(***W″*** is not the same matrix as ***W***.) If you want to learn more about dynamical systems, see Luenberger (1979).

Let's see how this might work.

Let **e**(t) be the input activity to the neurons, **f**(t) is the n-dimensional state vector representing output activity now as a

function of time. **W** is a fixed nxn weight matrix. The equation in the previous section is the steady state solution to the following differential equation:

$$\frac{d\mathbf{f}}{dt} = \mathbf{e}[t] + \mathbf{W}.\mathbf{f}[t] - \mathbf{f}[t] \qquad (9)$$

(You can see this by noting that as before, "steady state" just means that the values of **f(t)** are not changing with time, i.e. **df/dt = 0**). We are going to develop a solution to this set of equations using a discrete-time approximation.

The state vector **f** at time t+Δt ($\epsilon = \Delta t$) can be approximated as:

$$\mathbf{f}(t + \Delta t) \cong \mathbf{f}(t) + \varepsilon[\mathbf{e}(t) + \mathbf{W}.\mathbf{f}(t) - \mathbf{f}(t)]$$

We will fix or "clamp" the input **e**, start with arbitrary position of the state vector **f**, and model how the state vector evolves through time. We'll ask whether it seeks a stable state for which **f**(t) is no longer changing with time, **f**(t + Δt) = **f**(t),

i.e. when d**f**/dt = 0. In the limit as Δt (or ϵ) approaches zero, the solution is given by the steady state solution of the previous section. But neural systems take time to process their information and for the discrete time approximation, the system may not necessarily evolve to the steady state solution.

Simulation of recurrent lateral inhibition

First we will initialize parameters for the number of neurons (**size**), the space constant of the lateral inhibitory field (**spaceconstant**), the maximum strength of the inhibitory weights (**maxstrength**), the number of iterations (**iterations**), and the feedback delay **ϵ**:

■ The input stimulus

In[25]:=
```
size = 30;
spaceconstant =5;
maxstrength = 0.05;
iterations = 10;
ϵ = .3;
```

In[30]:=
```
e   = Join[Table[0,{i,N[size/3]}],Table[i/N[size/3],
        {i,N[size/3]}], Table[1,{i,N[size/3]}]];
g0  = ListPlot[e, PlotRange -> {{0,30},{-0.5,1.1}},PlotStyle->{RGBColor[1,0
picture = Table[e,{i,1,30}];
```

We've stored the graphic **g0** of the input for later use, we can show it later with **Show[g0]**.

■ Initializing the state vector and specifying the weights

Now we'll initialize the starting values of the output **f** to be random real numbers between 0 and 1, drawn from a uniform distribution.

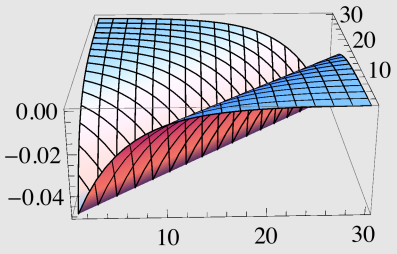
In[33]:=
```
f = RandomReal[{0, 1}, size];
```

Now let's set up synaptic weights which are negative, but become weaker the further they get from the neuron. We assume that the weights drop off exponentially away from each neuron:

In[39]:=
```
W =
Table[N[-maxstrength Exp[-Abs[i-j]/spaceconstant],1],
        {i,size},{j,size}];

ListPlot3D[W,ImageSize->Small]
```

Out[41]=



■ Simulating the response

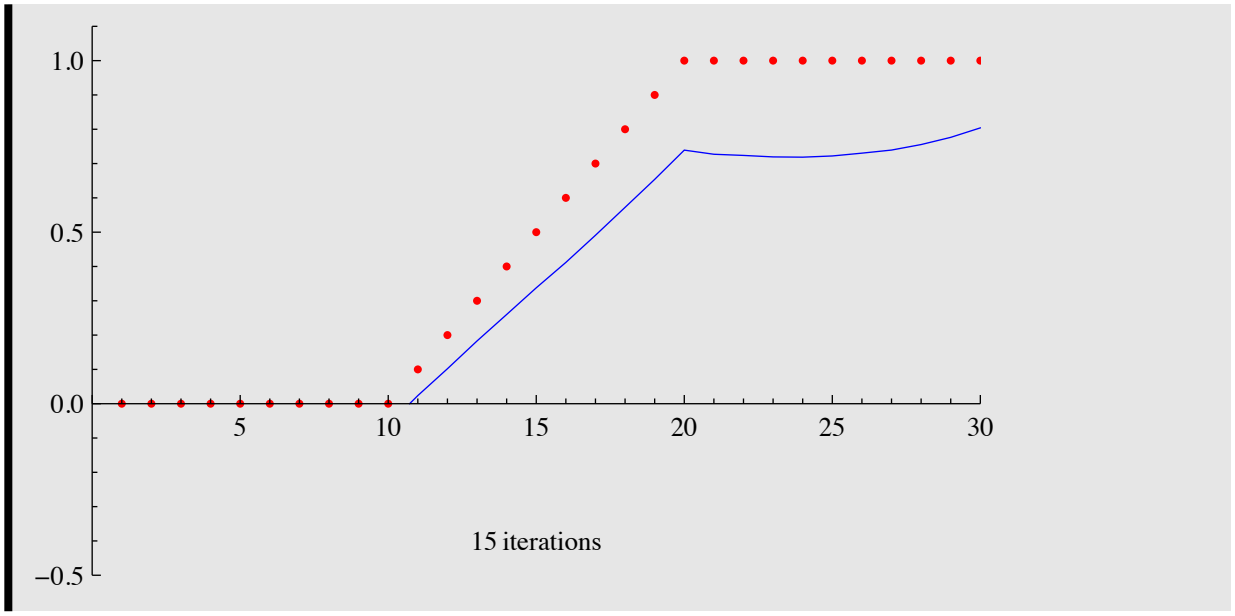
We are going to use the *Mathematica* function **Nest[ ]** to iterate through the limulus equations. `Nest[`*f*`,` *expr*`,` *n*`]` gives an expression with *f* applied *n* times to *expr*. For example, if we have defined a function **T[ ]**, **Nest[T,x,4]** produces as output **T[T[T[T[x]]]]**.

Let's express our discrete approximation for the limulus dynamical system in terms of a function, **T**, which will get applied repeatedly to itself with Nest:

In[42]:=
```
T[f_] := f + ϵ (e + W.f - f);
```

In[56]:=
```
iterations = 15;
g1 = ListPlot[Nest[T, f, iterations],PlotJoined->True,
        PlotRange -> {{0,30},{0,1.0}},PlotStyle->{RGBColor[0,0,1]}];
Show[g0,g1, Graphics[Text[iterations "iterations",
        {size/2,-0.4}]]]
```

Out[58]=



How does the simulation match up to data from the Limulus eye?

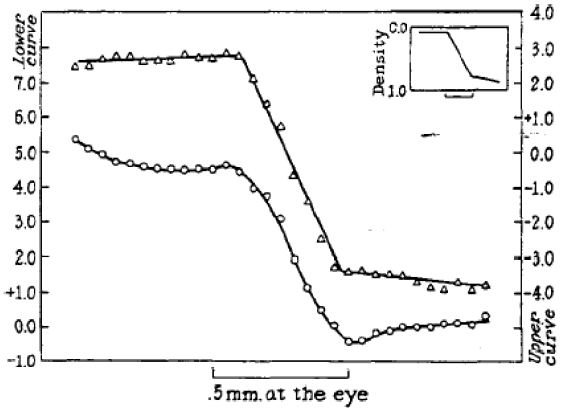


Fig. 8. Contrast phenomena, analogous to Mach bands, demonstrated by patterns of op - tic nerve fiber activity in the eye *of Limulus*. The discharge of impulses from a receptor was recorded as the eye was caused to scan slowly a pattern of illumination containing a simple gradient of intensity shown in the inset, upper right. When all the receptors were masked except the one from which activity was being recorded, a faithful representation of the actual physical distribution of light was obtained (upper graph, triangles). With the mask removed, so that all the receptors viewed the pattern, the lower graph (circles) was obtained, with a maximum and a minimum where Mach bands are seen by a human observer viewing the same pattern. (From Ratliff and Hartline[21])

From Hartline's Nobel lecture http://www.nobel.se/medicine/laureates/1967/hartline-lecture.pdf. Figure from: F. Ratliff and H.K. Hartline, *J. Gen. Physiol.*, 42 (1959) 1241.

Google limulus

Explore the parameter space

The effect of ϵ, strength of inhibition, and number of iterations

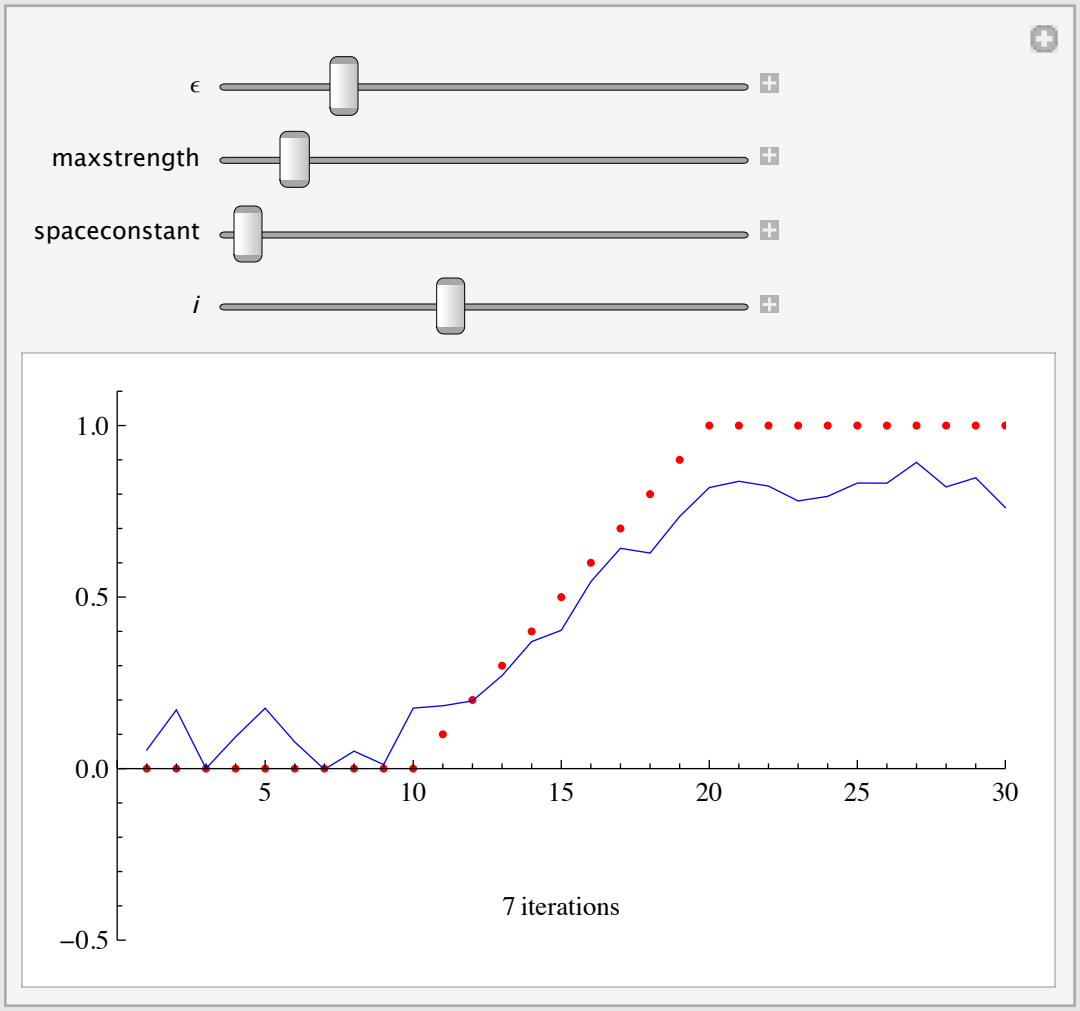
■ Define a function with inputs: ϵ, maxstrength and iterations, and outputs: a plot of response

We can use the **Module[ ]** function to define a routine with local variables and a set of other functions to define limulus[ϵ_,maxstrength_,iterations_]:

In[78]:=
```
limulus[ϵ_, maxstrength_, spaceconstant_, iterations_] :=
 Module[{f, W},
  W = Table[N[-maxstrength ℯ^(-Abs[i-j]/spaceconstant), 1], {i, size}, {j, size}];
  f = RandomReal[{0, 1}, size]; T[f_] := f + ϵ (e + W.f - f);
  g1 = ListPlot[Nest[T, f, iterations], Joined → True,
    PlotRange → {{0, 30}, {0, 1.`}}, PlotStyle → {RGBColor[0, 0, 1]}];
  Show[g0, g1, Graphics[Text[iterations "iterations", {size/2, -0.4}]]]]
```

In[81]:=
```
Manipulate[limulus[ϵ, maxstrength, spaceconstant, i], {{ϵ, .3}, 0, 1},
 {{maxstrength, .05}, 0, .5}, {{spaceconstant, 5}, 1, 15, 1},
 {i, 1, 15, 1}]
```

Out[81]=



What does the steady state response look like if the inhibition is small (i.e. small maxstrength)?

In[48]:=
```
limulus[.3, .005, 15];
```

What does the steady state response look like if the inhibition is large?

What does the steady state response look like if the spaceconstant is very small or very large?

What if the iteration step-size, ϵ, is large (e.g. 1.5). Run it limulus[ ] several times.

Neural networks as dynamical systems

We've explored a simple linear neural network that is a good model of limulus processing, and seems to provide a possible explanation for human perception of Mach bands. Real neural networks typically have non-linearities. There is no general theory of non-linear systems of difference or differential equations. But the exploration of this linear set does lead us to ask questions which are quite general about dynamical systems:

What does the trajectory in state-space look like?

Does it go to a stable point?

How many stable points or "attractors" are there?

There are non-linear systems which show more interesting behavior in which one sees:

Stable orbits

Chaotic trajectories in state-space

"Strange" attractors

We will return to some of these questions later when we study Hopfield networks.

Recurrent lateral inhibition & Winner-take-all (WTA)

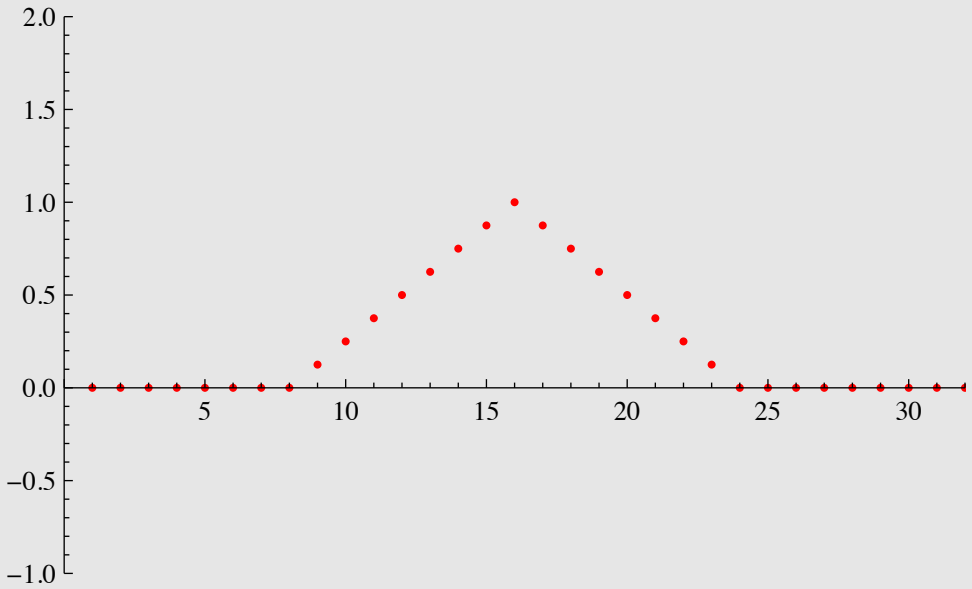
Sometimes one would like to have a network that takes in a range of inputs, but as output would like the neuron with biggest value to remain high, while all others are suppressed. In other words, we want the network to make a decision. The limulus equations can be set up to act as such a "winner-take-all" network. We will remove self-inhibition by setting all the diagonal elements of **W** to zero. We will also add a non-linear thresholding function ("rectification") to set negative values to zero, and we will increase the spatial extent of the inhibition.

■ Make a rectifying threshold function

```
thresh[x_] := N[If[x < 0.0, 0.0, x]];
SetAttributes[thresh, Listable];
```

■ **Make a "tepee" stimulus and initialize the neural starting values**

```
size = 32;
e   = Join[Table[0,{i,N[size/4]}],
        Table[i/N[size/4],{i,N[size/4]}],
        Table[(N[size/4]-i)/N[size/4],{i,N[size/4]}],
        Table[0,{i,N[size/4]}]];
g0  = ListPlot[e, PlotRange -> {{0,size},{-1,2.0}},PlotStyle→{RGBColor[1,0,
```

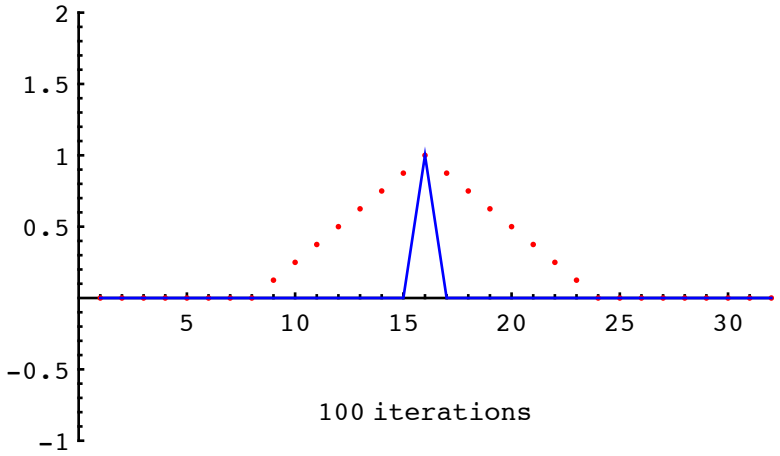


■ **Define winnertakeall[ ] as for limulus[ ], but with no self-inhibition:**

```
winnertakeall[ϵ_, maxstrength_, iterations_, spaceconstant_] :=
 Module[{f, W},
  W = Table[N[-maxstrength ℯ^(-Abs[i-j]/spaceconstant), 1], {i, size}, {j, size}];
  For[i = 1, i ≤ size, i++, W[[i, i]] = 0.`]; f = RandomReal[{0, 1}, size];
  T[f_] := thresh[f + ϵ (e + W.f - f)];
  g1 = ListPlot[Nest[T, f, iterations], Joined → True,
    PlotRange → {{0, size}, {-1, 2.`}}, PlotStyle → {RGBColor[0, 0, 1]}];
  Show[g0, g1, Graphics[Text[iterations "iterations", {size/2, -0.8`}]]]]
```

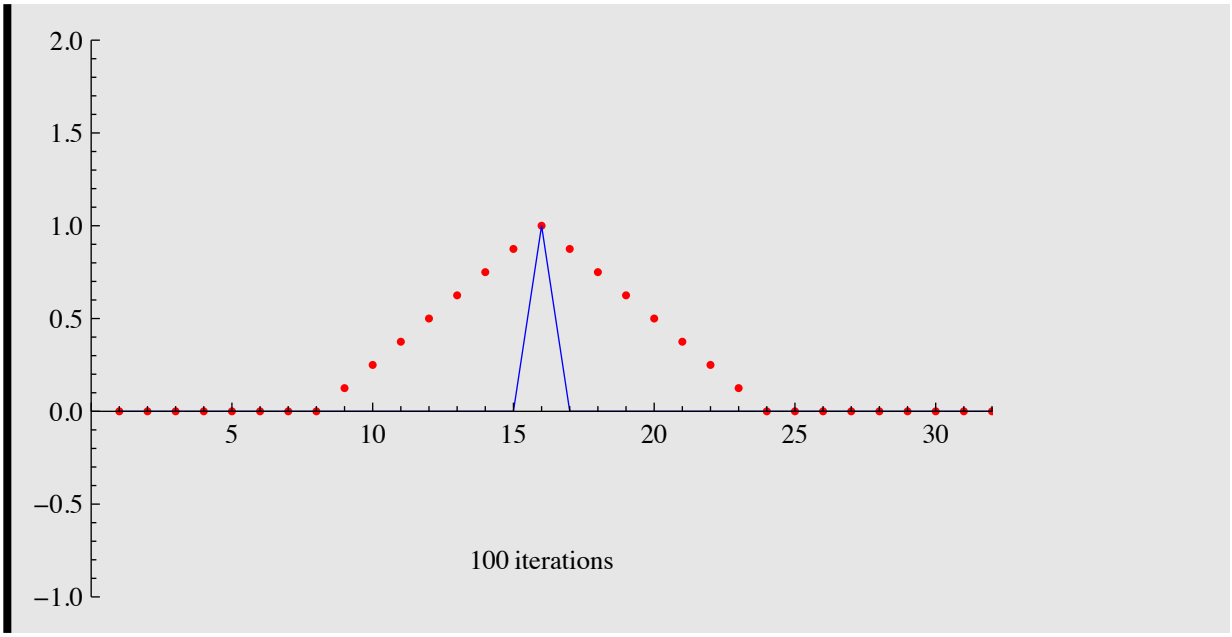
Use ListPlot3D[W] to see the modified structure of the weight matrix

Run simulation: Find a set of parameters that will select the maximum response and suppress the rest



If we think of the number of iterations to steady-state as "reaction time", how is this neural network for making decisions? How sensitive is its function to the choice of parameters?

If you are having a hard time finding a good set of parameters, select the cell below, then go to **Cell->Cell Properties->Cell Open**, and then run it.



Next time

- **Review matrices. Representations of neural network weights.**

References

Anderson, J. A. (1995). An Introduction to Neural Networks . Cambridge, MA: MIT Press. (Chapter 4.)

Boyaci, H., Fang, F., Murray, S. O., & Kersten, D. (2007). Responses to lightness variations in early human visual cortex. Curr Biol, 17(11), 989-993.

Hartline, H. K., & Knight, B. W., Jr. (1974). The processing of visual information in a simple retina. Ann N Y Acad Sci, 231(1), 12-8.

http://www.mbl.edu/animals/Limulus/vision/index.html

Knill, D. C., & Kersten, D. (1991). Apparent surface curvature affects lightness perception. Nature, 351, 228-230. (pdf) .http://gandalf.psych.umn.edu/~kersten/kersten-lab/demos/lightness.html

Luenberger, D.G. (1979). Introduction to dynamic systems : theory, models, and applications. (pp. xiv, 446). New York: Wiley.

Hartline, HK, Wagner, HG, & Ratliff , F. (1956) Inhibition in the Eye of Limulus, Journal of General Physiology, 39:5 pp.651-673

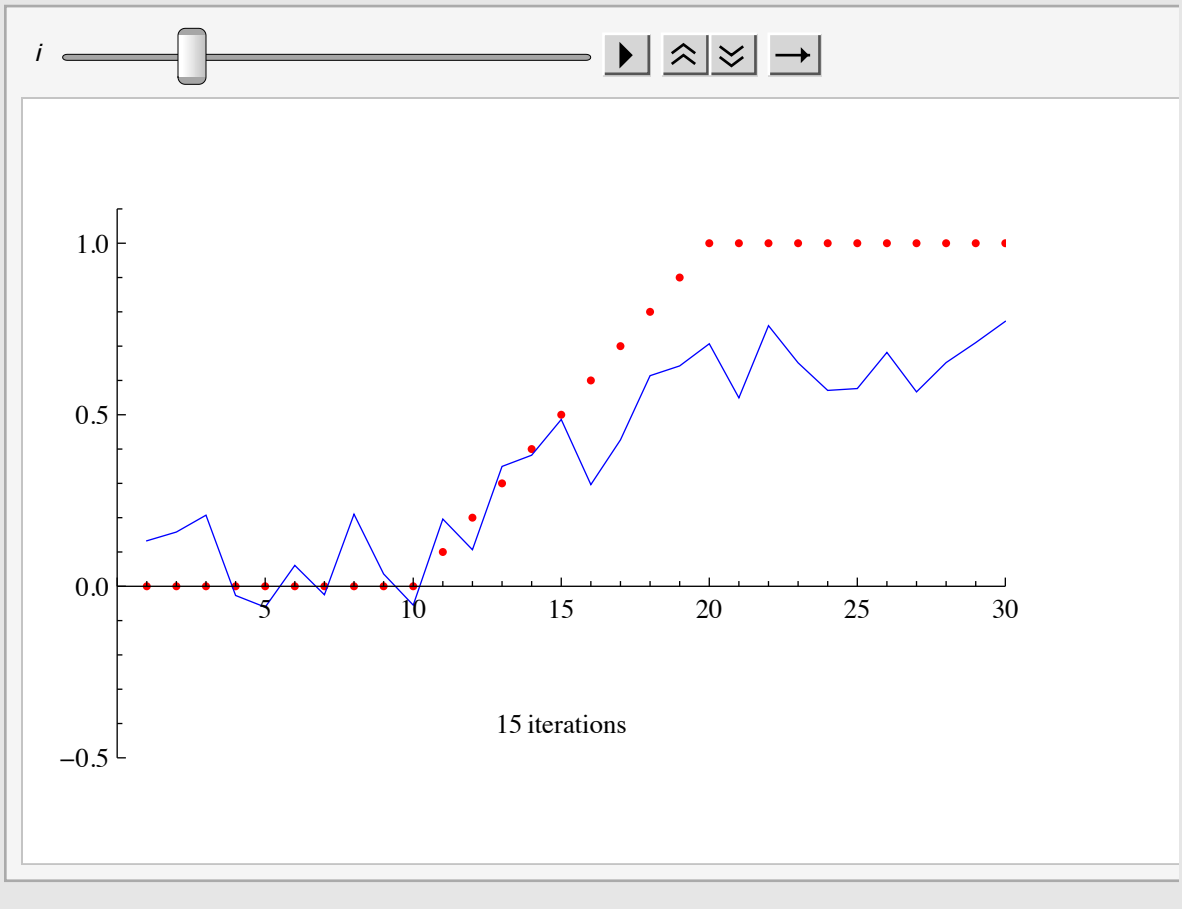
Ratliff, F., Knight, B. W., Jr., Dodge, F. A., Jr., & Hartline, H. K. (1974). Fourier analysis of dynamics of excitation and inhibition in the eye of Limulus: amplitude, phase and distance. Vision Res, 14(11), 1155-68.

Appendix

■ **Use NestList[ ] to store all the iterations before showing the results.**

In[72]:=
```
temp = NestList[T, f, 15];
Animate[
 Show[g0, ListPlot[temp[[i]], PlotJoined → True,
   PlotRange → {{0, 30}, {-.5, 1.0}}, PlotStyle → {RGBColor[0, 0, 1]}],
  Graphics[Text[iterations "iterations",
        {size / 2, -0.4}]]], {i, 1, 15, 1}]
```

Out[73]=

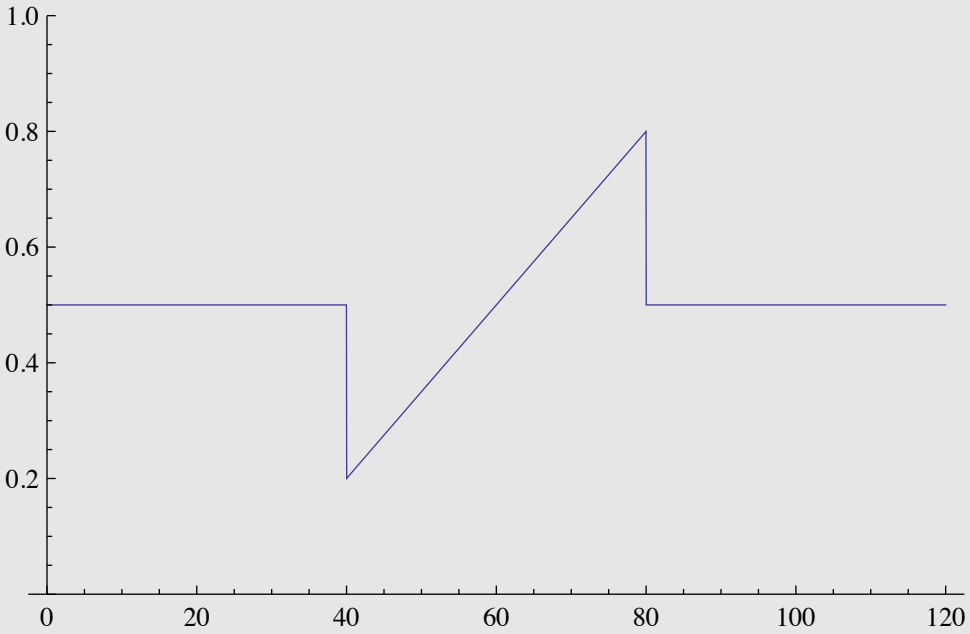


Exercise: Make a gray-level image of the horizontal luminance pattern shown below.

Does the left uniform gray appear to be the same lightness as the right patch? Can you explain what you see in terms of lateral inhibition?

```
low = 0.2; hi = 0.8;
left = 0.5; right = 0.5;
y2[x_] := left /; x<40
y2[x_] :=
    ((hi-low)/40) x + (low-(hi-low)) /; x>=40 && x<80
y2[x_] := right /; x>=80
```

```
Plot[y2[x], {x, 0, 120}, PlotRange → {0, 1}]
```



Exercise: Hermann grid

Below is the Hermann Grid. Notice the phantom dark spots where the white lines cross. Can you explain what you see in terms of lateral inhibition?

```
width2 = 5; gap = 1; nsquares = 6;
```

```
hermann = Flatten[Table[{Rectangle[{x, y}, {x + width2, y + width2}]},
{x, 0, (width2 + gap) * (nsquares - 1), width2 + gap},
    {y, 0, (width2 + gap) * (nsquares - 1), width2 + gap}], 1];
```

```
Show[Graphics[hermann, AspectRatio → 1]]
```