Modeling Receptive Fields as Linear Filters in Space and Time

In this tutorial, we will build a model of the firing rate responses of a neuron in the lateral geniculate nucleus (LGN) to a stimulus that varies over space and time. We will then apply this model to understand the neural basis of a visual illusion known as "forward masking" in which a target stimulus is made to appear less visible by a previously presented "masking" stimulus.

I. Background: LGN responses in space and time

Neurons in the LGN respond optimally to spatial patterns consisting of dots of one contrast (light or dark) surrounded by an annulus (i.e. ring) of the opposite contrast (dark or light, respectively). We model this spatial dependence of an LGN neuron's firing rate by a correlation integral between the stimulus light intensity s(x) and the cell's spatial receptive field D(x):

$$r = \left[r_0 + \int dx D(x) s(x) \right]_+ , \tag{1}$$

where r_0 gives the background rate in the absence of input, the integral is over all spatial locations, and D(x) determines how much weight is given to each spatial location at which a stimulus might be presented. $[\ ]_+$ indicates thresholding. For simplicity, we consider only one spatial dimension in the above expression and in the discussion below.

Typically, D(x) for retinal cells and for LGN cells is modeled as a difference of Gaussian (i.e. bell curve) functions:

$$D(x) = \frac{A_c}{\sqrt{2\pi\sigma_c^2}} \exp\left(-\frac{x^2}{2\sigma_c^2} \right) - \frac{A_s}{\sqrt{2\pi\sigma_s^2}} \exp\left(-\frac{x^2}{2\sigma_s^2} \right) \tag{2}$$

where the first term gives the excitatory center portion of the response and the second term gives the inhibitory surround portion of the response. A_c determines the strength (technically, the area under the Gaussian function curve) and σ_c determines the width of the excitatory center portion of the receptive field. A_s determines the strength and σ_s, which is greater than σ_c, the spatial extent of the inhibitory surround portion of the receptive field.

The above expression for the firing rate gives the response of the neuron to a constant (non-varying in time) stimulus. Conversely, we might want to consider the response in time to a stimulus s(t) that is uniform in space across the entire field of view but varies in time. This temporal response is characterized by a temporal receptive field function (or "kernel") $D(\tau)$ that characterizes how much influence the stimulus a time τ ago has on the current firing rate:

$$r(t) = \left[r_0 + \int d\tau D(\tau) s(t-\tau) \right]_+ , \tag{3}$$

where the integral goes from $\tau = 0$ to $\tau = \infty$. Note in the above expression that $D(\tau)$ determines how much weight is given to stimuli presented at time $t-\tau$, or a time τ before the present time t.

$D(\tau)$ for LGN cells is, similar to the spatial receptive field D(x), given by a difference between a shorter positive portion and a longer negative part. It is often given by (e.g. see Dayan and Abbott, p. 66) a function of the form:

$$D(\tau) = \alpha \exp(-\alpha\tau) \left(\frac{(\alpha\tau)^5}{5!} - \frac{(\alpha\tau)^7}{7!} \right) \tag{4}$$

where n! = "n factorial" = n*(n-1)*(n-2)*...*1. α here determines how elongated the temporal kernel is (with smaller α corresponding to longer kernels).

Finally, for stimuli s(x,t) that are varying in both space and time we must consider the full spatio-temporal receptive field D(x,τ) that determines the amount by which a stimulus located at position x a time τ ago contributes to the firing rate of the neuron. This gives the full expression for the firing rate as an integral over all spatial locations and all previous times:

$$r(t) = \left[r_0 + \int dx \int d\tau D(x,\tau) s(x, t-\tau) \right]_+, \quad (5)$$

where the limits of the integrals are as in the purely spatial and purely temporal cases considered above.

Often it is the case that the kernel D(x,τ) is in the form of a product $D(x,\tau)=D_x(x)D_t(\tau)$ where $D_x(x)$ is the spatial portion of the kernel (given in our case by equation (2)) and $D_t(\tau)$ is the temporal portion of the kernel (given in our case by equation (4)). [*Notation note: we use "D" as a general notation for kernels. The subscript 'x' or 't' then is used to be more specific about whether we mean the spatial or temporal kernel, and should not be confused with the arguments received by the function, which are in parentheses.*] When D(x,τ) can be written in the form of a product of separate spatial and temporal kernels, it is called *separable* in space and time. This means that the shape of the temporal preference of the neuron, given by $D_t(\tau)$, is independent of (and thus can be considered separately from) the point x in space that is being considered (and vice-versa: the spatial receptive field for such a neuron is independent of the time at which this pattern appears). We will gain intuition for this concept later in the tutorial when we plot such a function graphically. When the space-time receptive field D(x,τ) cannot be written in the form of such a product it is called *inseparable*.

For the LGN neurons considered below, we will assume that D(x,τ) is separable, which is a very reasonable assumption for such cells. Then we can re-write the integral in equation (5) as:

$$r(t) = \left[r_0 + \int dx D_x(x) \int d\tau D_t(\tau) s(x, t-\tau) \right]_+ \quad (6)$$

If, moreover, the stimulus can be written as a separable product $s(x,t)=s_x(x)s_t(t)$, then the integral simplifies even more:

$$r(t) = \left[r_0 + \left(\int dx D_x(x) s_x(x) \right) \left(\int d\tau D_t(\tau) s_t(t-\tau) \right) \right]_+ \quad (7)$$

Notice that, in this special case where both the stimulus and receptive field are separable, the double integral factors into a product of two entirely independent integrals representing the spatial response and temporal response, respectively. This case is particularly simple and fast to compute.

Finally, note that all of the equations for r(t) in this section consist of a background piece plus an integral that corresponds to multiplying each "space-time pixel" of the stimulus s(x,t-τ) by a corresponding weighting function D(x,τ). Because we can think of the integral as simply adding up all of these contributions, we ultimately get that (before thresholding) r(t) is just a weighted sum, or *linear combination*, of all of the stimulus values preceding it. This integral is therefore called a *"linear filter"* of the stimulus s(x,t), and D(x,τ) itself is sometimes called the filter function. The threshold is then applied to the result of the linear filtering (people consider the constant background rate to be part of the linear filter, just like the equation of a line y=mx + b has a constant piece b plus a piece that depends linearly on x). Since the thresholding is a nonlinear function (equal to zero for negative values of its argument but equal to multiplying by 1 for positive values), but also does not change in time (i.e. is *static* in time), it is called a *static nonlinearity*. Sometimes modelers use more complicated static nonlinearity functions instead of simply thresholding, but thresholding is the most common choice of nonlinearity function.

II. Plotting the receptive fields

In the following, we will be seeing how the above-described LGN receptive field model can be used to model the phenomenon of forward masking. We will show that, whereas a neuron responds strongly to a target bar of light presented to it in isolation, the neuron may not respond to this same target bar of light if another bar (or bars) of light (known as the "mask") is presented to it first. We will explore where in space and when in time the masking bar(s) of light need to be presented in order to cause the masking to occur.

In this section, we'll take a look at our receptive field $D(x,\tau)=D_x(x)D_t(\tau)$ by plotting it as a function of space and of time. Then, in the next sections of this tutorial, we will add a stimulus and see how the firing rate r(t) changes over time in response to this stimulus.

Open up a new file and save it with a useful name like "ForwardMasking.m". Then add some descriptive information at the top and clear all and close all, as usual:

```
%model an LGN neuron receiving a target bar of light preceded by two
%masking bars of light that, if appropriately positioned in space and
%appropriately timed, will make the target bar appear less visible!

clear all;
close all;
```

Let's first plot the spatial receptive field $D_x(x)$. To start, let's write down the relevant parameters:

```
%FILTER PARAMETERS
%spatial filter parameters
sigma_c = 1.4;  %width of center portion of spatial r.f. [degrees]
sigma_s = 2.1;  %determines width of surround portion of spatial r.f. [deg]
A_c = 1;  %strength of center portion of spatial r.f.
A_s = 0.9; %strength of surround portion of spatial r.f.
dx = 0.05; %resolution of spatial grid
```

This should give the receptive field as a difference between a center Gaussian of area $A_c = 1$ and a surround Gaussian of area $A_s = 0.9$, so that for this cell the center is stronger than the surround (i.e. presenting a stimulus that is uniform in value across the entire receptive field would excite the cell). Note that image sizes are usually measured in terms of the angle the image subtends on the eye (or correspondingly the angular size of the image if we place the origin at the location of the pupil).

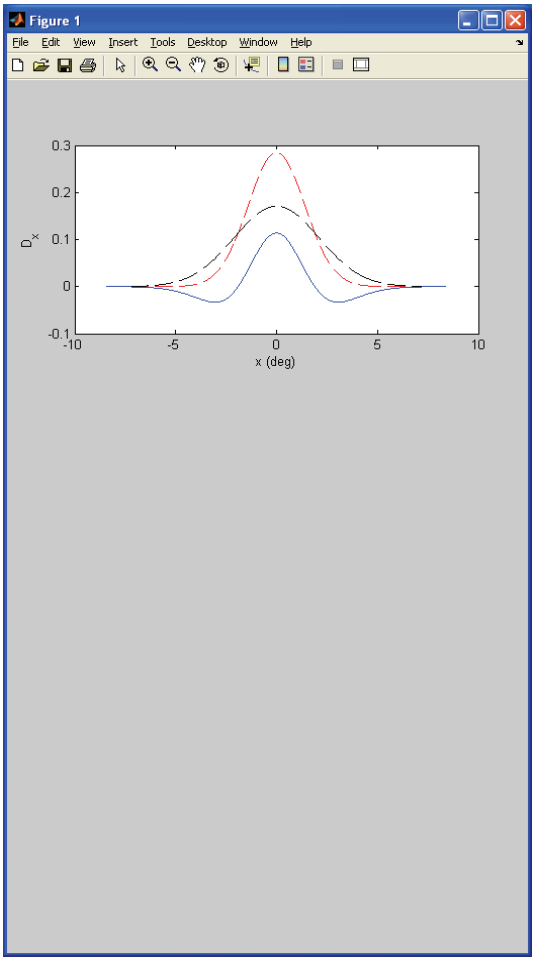
Next let's define and then plot the receptive field. We really only need to plot it for values significantly greater than zero, so let's plot over a range of x going out to $\pm 4*\sigma_s$ (note: this means $\pm$ 4 standard deviations of the Bell curve). To do this, next add:

```
x_min = -4*sigma_s; %x-value roughly giving -x border of r.f.
x_max = 4*sigma_s;  %x-value roughly giving +x border of r.f.
x_vect = x_min:dx:x_max; %values of x over which to compute D_x
```

Then define $D_x(x)$ according to equation (2). To do this, let's define and plot the center Gaussian and surround Gaussian separately as well as the full spatial receptive field which is the difference between these Gaussian functions:

```
%define and then plot spatial kernel D_x
D_x_center = A_c*exp(-(x_vect.^2)/(2*(sigma_c^2)))/sqrt(2*pi*sigma_c^2);
D_x_surround = A_s*exp(-(x_vect.^2)/(2*(sigma_s^2)))/sqrt(2*pi*sigma_s^2);
D_x_vect = D_x_center - D_x_surround;
figure(1)
subplot(3,1,1)
plot(x_vect, D_x_center, 'r--') %plot center with red dashed lines
hold on
plot(x_vect, D_x_surround, 'k--') %plot surround with black dashed lines
plot(x_vect, D_x_vect)  %plot full receptive field with solid blue line
xlabel('x (deg)')
ylabel('D_x')
```

Go ahead and run this now—you should see the following which illustrates that the receptive field is made of the difference of the two Gaussian functions shown in red and black:



Note that we have set up 3 subplots (even though only 1 is filled in now) because we will next plot $D_t(\tau)$ in the second subplot and then the full kernel $D(x,\tau)=D_x(x)D_t(\tau)$ in the third subplot.

For the temporal kernel $D_t(\tau)$, let's similarly add a vector for the times τ at which we want to compute the kernel. Add beneath the spatial filters section of the code:

```
%temporal filter parameters
alpha = 1/10; %determines length of temporal filter [ms^-1]
dt = 1; %ms
tau_vect_max = 20/alpha; %tau value giving border of temporal r.f.
tau_vect = 0:dt:tau_vect_max; %value of tau over which to compute D_t
```

Here, tau_vect_max is defined so that $D_t(\tau)$ will have decayed to zero by this time.

Next add (at the end of your current code) the code to compute and then plot $D_t(\tau)$ according to equation (4):

```
%define and then plot temporal kernel D_t
D_t_vect = alpha*exp(-alpha*tau_vect).*((alpha*tau_vect).^5/(5*4*3*2) - ...
            (alpha*tau_vect).^7/(7*6*5*4*3*2));
subplot(3,1,2)
plot(tau_vect,D_t_vect)
xlabel('tau (ms)')
ylabel('D_t')
```

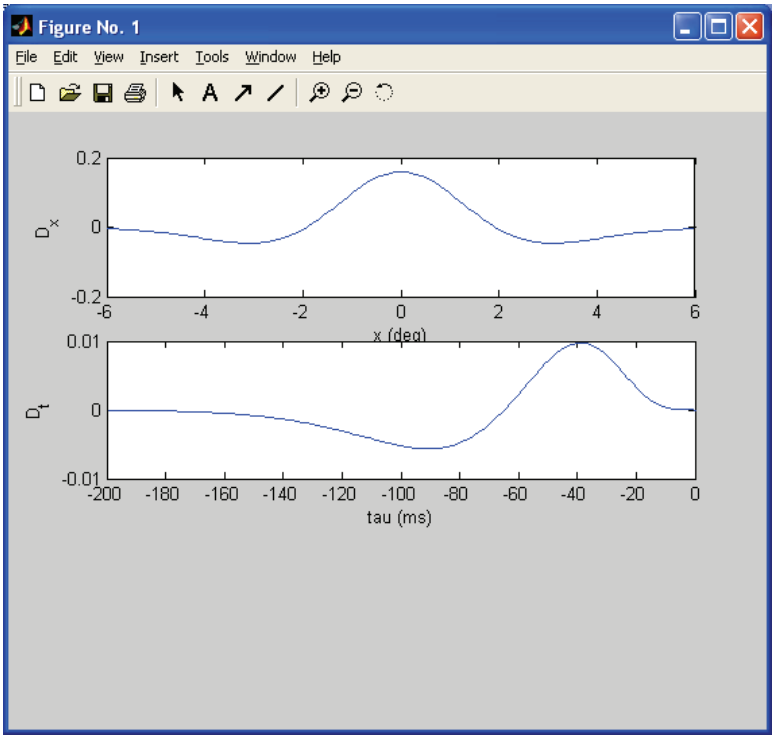
Go ahead and run this code now.

It is often conventional to plot temporal kernels with time going backwards (i.e. increasing to the left). To do this, add the following line following the plot command:

```
set(gca,'XDir','reverse')
```

The **set** command is used to set various parameters related to items like axes and figure traces. The argument "**gca"** means "get current axis" which tells MATLAB to access the set of axes into which data is currently set up to be plotted. The next two arguments say that the "X-Direction" of the x-axis should be plotted with positive values to the left (or "reversed").

Running this you should see (note below that I have commented out the plot commands for the dashed lines, in order to make the spatial receptive field plot less cluttered; you can do this as well):



Finally, let's compute the full kernel as the matrix $D(x,\tau)=D_x(x)D_t(\tau)$. This can be represented by a matrix with the first dimension corresponding to the x-values and the second dimension to the τ values, and can be attained simply in MATLAB by performing an "outer product" of the D_x and D_t vectors as follows:

```
%define and then plot the full spatio-temporal kernel D(x,tau)
D_xt_mat = D_x_vect'*D_t_vect; %full 2-D r.f., with x as 1st dimension & t as 2nd dimension
```

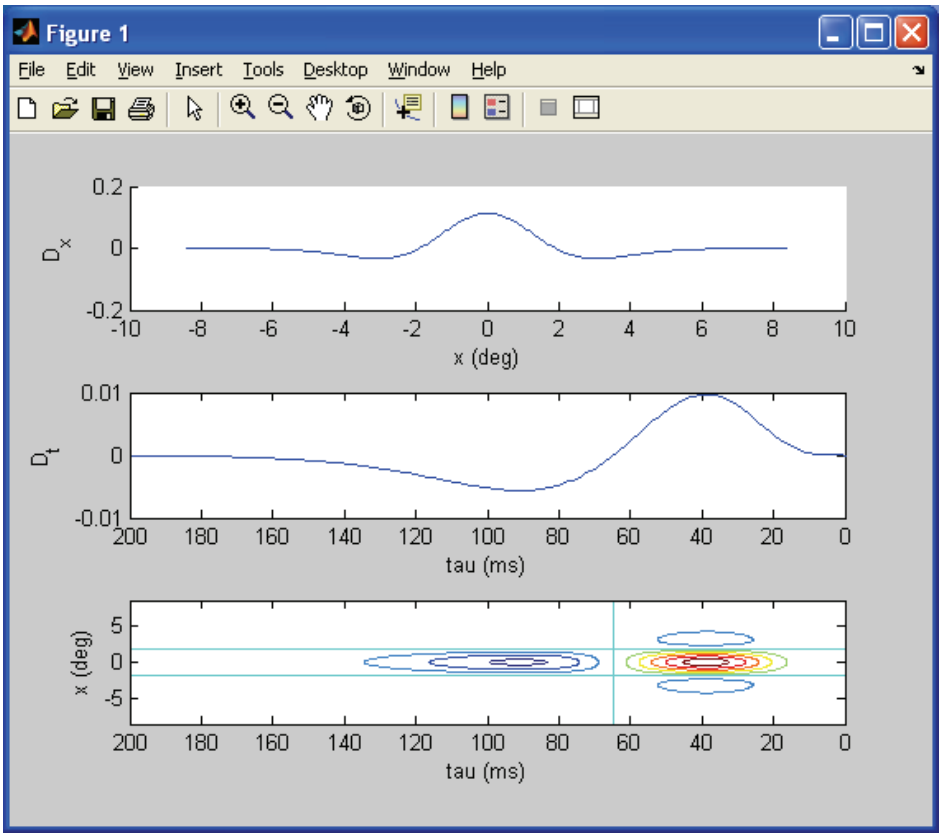
Note that the ' indicates that we transpose (turn row into column or vice-versa) the first vector and then do matrix multiplication as we learned about in the MATLAB tutorial (please brush up on this if it has faded from memory, e.g. by taking out a piece of paper and trying this operation for two short vectors of your choosing, and then checking your work at the MATLAB command line). You should find that the size of the resulting `D_xt_mat` matrix is 337 (the length of the D_x_vect) x 201 (the length of the D_t_vect).

To plot this function of 3 variables, we really need some sort of plot that captures 3-D information (i.e. the value of D on the z axis vs. the values of x and t on the other two axes). One nice way to capture such information is in a *contour plot*. Try typing:

```
subplot(3,1,3)
contour(tau_vect,x_vect,D_xt_mat); %makes a contour plot of the data
set(gca,'Xdir','reverse')
xlabel('tau (ms)')
ylabel('x (deg)')
```

The **contour(*x*,*y*,*z*)** command plots the function z = D_xt_mat as a function of its two dimensions (in our case, *x*=tau and *y*=x, where *x* and *y* here refer to what is being plotted on the

x- and y-axes respectively). Doing this you should now see something similar to the following (there may be more or fewer contours in the 3rd panel depending on the version of MATLAB you are running):

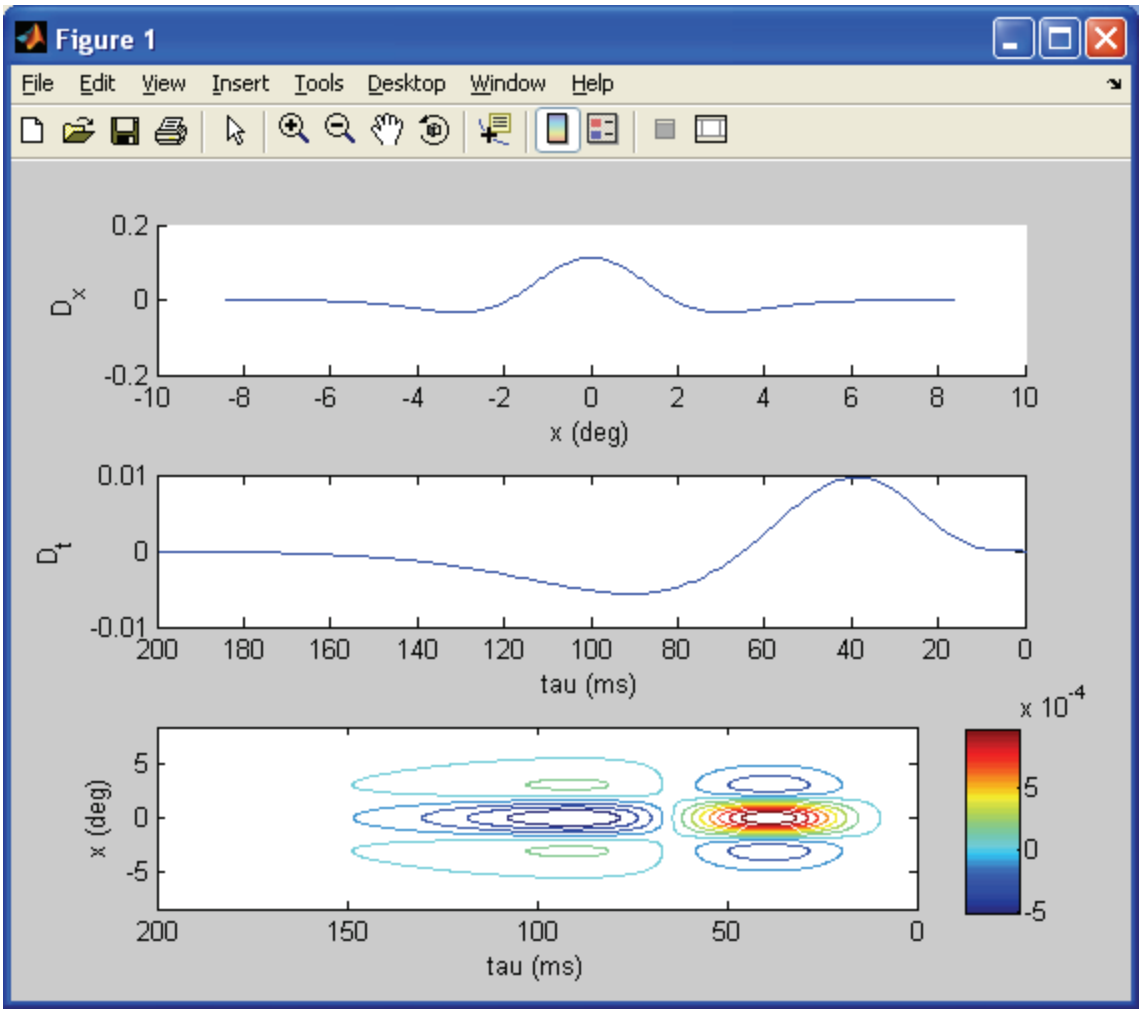


The contour plot shows the 2-D kernel, with more blue colors corresponding to more negative values and more red colors to more positive values. The way to read this is as follows: The contour lines give values of (x,τ) for which $D(x,\tau)$ has the same value. (It's called a 'contour' plot because this is like reading a map of geographical elevation where the points along a given contour each correspond to the same height above ground. In our case, each contour corresponds to the same z-value/height $D(x,\tau)$). To get the value of the kernel at a particular location x and particular time ago τ, look at the color of the corresponding point and, if it's white, extrapolate between the colors of the two neighboring contour lines.

If it seems that there aren't enough contours, we can specify this by adding a number to the end of the contour command. Furthermore, we can add a legend by writing the command **colorbar**. Try this by replacing your contour line by the two lines:

```
contour(tau_vect,x_vect,D_xt_mat,12); %makes a contour plot of the data
colorbar
```

This should produce 12 contours and also add a colorbar so you can know what values of $D(x,\tau)$ the particular colors correspond to. You should now see:



Now let's continue interpreting this. For example, if we would like to know the temporal kernel (i.e. weighting of previous times) for the point x=0, we need only follow the horizontal line of x = 0. This shows what we expect: going from right to left, the cell prefers positive (i.e. 'light') stimulus values for recent times in the past (with a maximum around -40 ms), preceded by negative (= 'dark') stimulus values for earlier times (with a minimum around -90 ms). Thus, since this is an ON-center cell, its response to a stimulus at x=0 would be strongest if the stimulus were to transition from dark to light in its center. Similarly, for a stimulus in the 'surround' (|x| greater than about 2 deg in our case), the preferred stimulus would be a transition from light to dark.

Note that for any value of x, the temporal pattern that the cell prefers is always the same (up to a minus sign) and in particular is equal to $D_t(\tau)$ (compare the 2nd subplot and a horizontal cross-section of the 3rd subplot for any given value of x). Mathematically, this shows that the overall receptive field $D(x,\tau)$ shown in subplot 3 temporal pattern is given by the temporal pattern of $D_t(\tau)$ shown in subplot 2 multiplied by the value of $D_x(x)$ for the corresponding x location. Thus, for example, the cell is most responsive to dark-followed-by-light at x=0 and responds most strongly to light-followed-by-dark (with the same temporal pattern) for values of x for which $D_x(x)$ is negative. The fact that the temporal pattern the cell prefers is the same (up to a minus

sign) for all positions x is a reflection of the *space-time separability* of this receptive field. To imagine a non-space-time separable field, imagine that the receptive field in subplot 3 were rotated counter-clockwise by, e.g., 45^0 so that horizontal sections through it at different spatial locations did not have the same temporal pattern.

One can similarly consider a fixed time back from the current time, e.g. $\tau = 50$ ms, and consider what spatial pattern of light and dark the cell responds to. In this case, at $\tau = 50$ ms, the cell prefers light in the center x locations and dark in the surround. At $\tau = 100$ ms, the cell prefers dark in the center and light in the surround (but again the same spatial pattern, namely $D_x(x)$, up to a minus sign).

Ok. Let's save this if you haven't already and move on to considering the response to a stimulus.

Your code at this point should read:

```
%model an LGN neuron receiving a target bar of light preceded by two
%masking bars of light that, if appropriately positioned in space and
%appropriately timed, will make the target bar appear less visible!

clear all;
close all;

%FILTER PARAMETERS
%spatial filter parameters
sigma_c = 1.4;  %width of center portion of spatial r.f. [degrees]
sigma_s = 2.1;  %determines width of surround portion of spatial r.f. [deg]
A_c = 1;  %strength of center portion of spatial r.f.
A_s = 0.9; %strength of surround portion of spatial r.f.
dx = 0.05; %resolution of spatial grid
x_min = -4*sigma_s; %x-value roughly giving -x border of r.f.
x_max = 4*sigma_s;  %x-value roughly giving +x border of r.f.
x_vect = x_min:dx:x_max; %values of x over which to compute D_x

%temporal filter parameters
alpha = 1/10; %determines length of temporal filter [ms^-1]
dt = 1; %ms
tau_vect_max = 20/alpha; %tau value giving border of temporal r.f.
tau_vect = 0:dt:tau_vect_max; %value of tau over which to compute D_t

%define and then plot spatial kernel D_x
D_x_center = A_c*exp(-(x_vect.^2)/(2*(sigma_c^2)))/sqrt(2*pi*sigma_c^2);
D_x_surround = A_s*exp(-(x_vect.^2)/(2*(sigma_s^2)))/sqrt(2*pi*sigma_s^2);
D_x_vect = D_x_center - D_x_surround;
figure(1)
subplot(3,1,1)
%plot(x_vect, D_x_center, 'r--') %plot center with red dashed lines
hold on
%plot(x_vect, D_x_surround, 'k--') %plot surround with black dashed lines
plot(x_vect, D_x_vect)  %plot full receptive field with solid blue line
xlabel('x (deg)')
ylabel('D_x')
```

```
%define and then plot temporal kernel D_t
D_t_vect = alpha*exp(-alpha*tau_vect).*((alpha*tau_vect).^5/(5*4*3*2) - ...
        (alpha*tau_vect).^7/(7*6*5*4*3*2));
subplot(3,1,2)
plot(tau_vect,D_t_vect)
set(gca,'XDir','reverse')
xlabel('tau (ms)')
ylabel('D_t')

%define and then plot the full spatio-temporal kernel D(x,tau)
D_xt_mat = D_x_vect'*D_t_vect; %full 2-D r.f., with x as 1st dimension
                        %& t as 2nd dimension
subplot(3,1,3)
contour(tau_vect,x_vect,D_xt_mat,12); %makes a contour plot of the data
colorbar
set(gca,'Xdir','reverse')
xlabel('tau (ms)')
ylabel('x (deg)')
```

III. Add and plot a stimulus (the Target bar) in space and time

For the stimulus, at any given time, we will either be presenting a bar of light flanked on either side by two dark patches (= the Target) or the reverse of this (i.e. two bars of light in the flanks with a dark bar in the center). Let's assign a stimulus value of 1 to light points and 0 to dark points.

First let's test the response to just a bar of light flashed for a fixed amount of time. To parameterize this, let's make a new section of our code and first define our spatial parameters:

```
%STIMULUS PARAMETERS
%spatial parameters & plot of spatial locations of stimuli
Target_LeftEnd = -0.5; %position of start of bar [deg]
Target_RightEnd = 0.5; %position of end of bar [deg]
Target_x_vect = [zeros(1,(Target_LeftEnd - x_min)/dx)... %nothing flashed here
                 ones(1,((Target_RightEnd - Target_LeftEnd)/dx)+1)... %Target position
                 zeros(1,(x_max - Target_RightEnd)/dx)]; %nothing flashed here
```

The first two lines indicate that the Target bar will extend from -0.5^0 to $+0.5^0$. The final line then defines the target as being light (=1) where it is located and 0 (= dark) where it is not.

Next let's plot this Target in a new figure that will also hold 3 subplots (for stimulus as a function of space, time, and space & time together):

```
figure(2)
subplot(3,1,1)
plot(x_vect,Target_x_vect,'o')
xlabel('x (deg)')
ylabel('Target(1=Lt,0=Dk)')
```

Try running this to see if you get a target bar. Next let's define when, in time, the stimulus is on and then plot this. The code is analogous to the spatial code, except that we added a t_vect to hold the times for which the simulation is run:

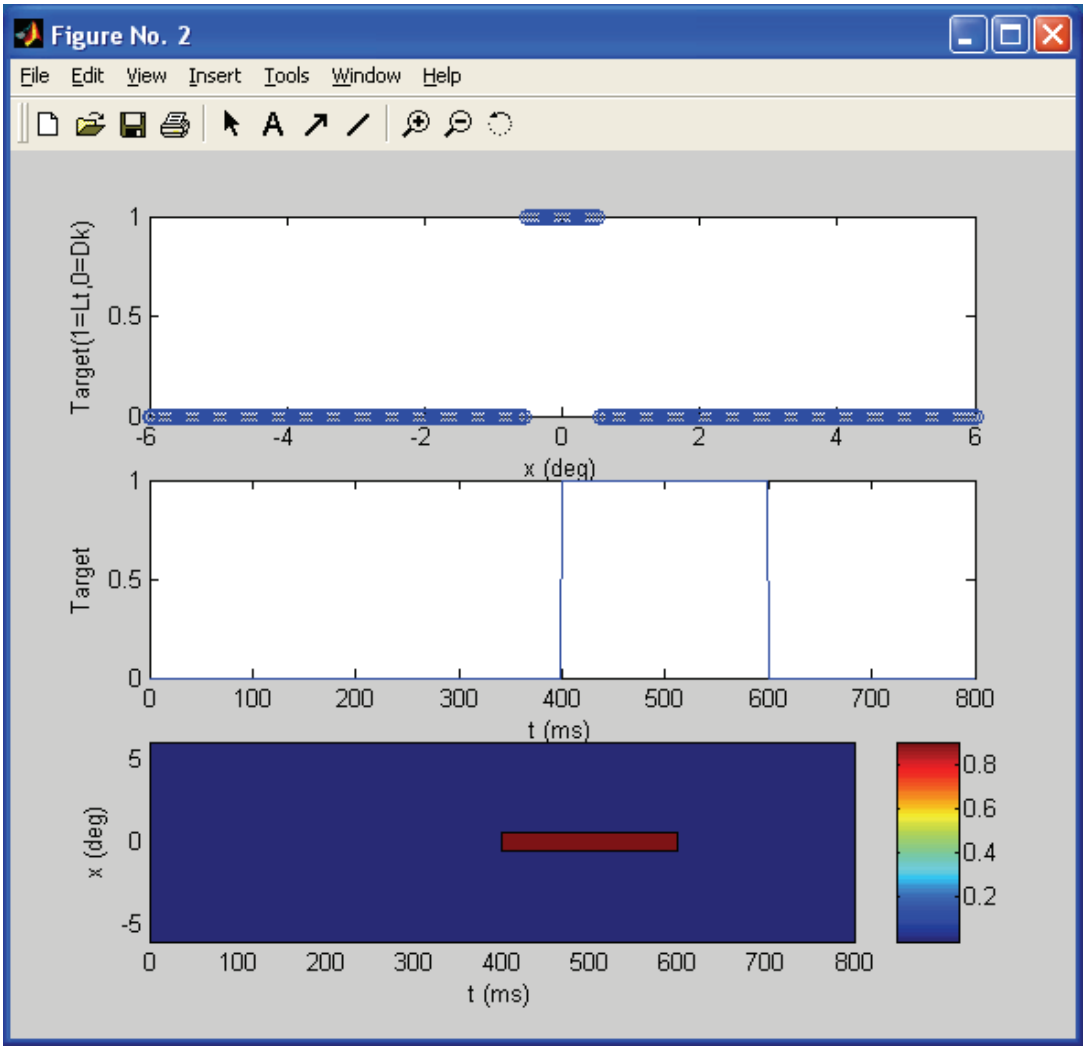
```
%temporal parameters & plot of temporal locations of stimuli
tmax = 800; %ms
Target_on = 400; %ms
Target_off = 600; %ms
t_vect = 0:dt:tmax;
Target_t_vect = [zeros(1,Target_on/dt) ...   %bar initially off from t=0 to t=Target_on-dt
            ones(1,(Target_off-Target_on)/dt) ... %then Target on
            zeros(1,((tmax-Target_off)/dt)+1)];  %then Target off again
subplot(3,1,2)
plot(t_vect,Target_t_vect)
xlabel('t (ms)')
ylabel('Target')
```

Try running this. Now, note that if we want the stimulus over space and time, our convenient choice of dark = 0 means that the stimulus is separable in space and time, i.e. s(x,t) = s(x)*s(t). This product is zero whenever s(t)=0, i.e. when the stimulus is turned off. Further, the product correctly equals s(x) when the stimulus is on because s(t)=1 at these times. Thus at all times the stimulus equals s(x)*s(t). Let's try plotting this, using a contour plot:

```
%define and plot stimulus in both space & time
Target_xt_mat = Target_x_vect'*Target_t_vect;
subplot(3,1,3)
contourf(t_vect,x_vect,Target_xt_mat); %makes a contour plot of the data
colorbar
xlabel('t (ms)')
ylabel('x (deg)')
```

Now run this. In this case, the contour plot shows the location of the stimulus in both space and time (note that we run time 'forward', i.e. positive, for the stimulus as usual. It is only the temporal kernel that we plot 'backwards' to indicate that the kernel values weigh points earlier in time). However, it would be nicer in this case if the Target and non-target areas were filled with a solid color rather than being empty. We can do this by replacing the contour command by **contourf** which stands for "filled" (with color) contour.

Do this and you should see the following:



Now the times at which the target is on are filled red and those at which the target is off are blue (the thin black border is the transition region between neighboring spatial or temporal steps, whose width reflects the value of dx in space and dt in time).

If you prefer it, you can also go back and change your figure(1) plot to a filled contour plot.

Your code for this section should be:

```
%STIMULUS PARAMETERS
%spatial parameters & plot of spatial locations of stimuli
Target_LeftEnd = -0.5; %position of start of bar [deg]
Target_RightEnd = 0.5; %position of end of bar [deg]
Target_x_vect = [zeros(1,(Target_LeftEnd - x_min)/dx)... %nothing flashed here
                 ones(1,((Target_RightEnd - Target_LeftEnd)/dx)+1)... %Target position
                 zeros(1,(x_max - Target_RightEnd)/dx)]; %nothing flashed here
figure(2)
subplot(3,1,1)
plot(x_vect,Target_x_vect,'o')
xlabel('x (deg)')
```

```
ylabel('Target(1=Lt,0=Dk)')
%temporal parameters & plot of temporal locations of stimuli
tmax = 800; %ms
Target_on = 400; %ms
Target_off = 600; %ms
t_vect = 0:dt:tmax;
Target_t_vect = [zeros(1,Target_on/dt) ...   %bar initially off from t=0 to t=Target_on-dt
            ones(1,(Target_off-Target_on)/dt) ... %then Target on
            zeros(1,((tmax-Target_off)/dt)+1)];  %then Target off again
subplot(3,1,2)
plot(t_vect,Target_t_vect)
xlabel('t (ms)')
ylabel('Target')
%define and plot stimulus in both space & time
Target_xt_mat = Target_x_vect'*Target_t_vect;
subplot(3,1,3)
contourf(t_vect,x_vect,Target_xt_mat); %makes a contour plot of the data
colorbar
xlabel('t (ms)')
ylabel('x (deg)')
```

IV. Plot the rate as a function of space and time for the Target stimulus

We now have both a space-time separable kernel function $D(x,\tau)=D_x(x)D_t(\tau)$ and a space-time separable stimulus $s(x,t)=s_x(x)s_t(t)$ where in this case the stimulus is the Target. Because of this space-time separability of both kernel and stimulus, we can use equation (7):

$$r(t) = \left[r_0 + \left(\int dx D_x(x) s_x(x) \right) \left(\int d\tau D_t(\tau) s_t(t-\tau) \right) \right]_+$$
$$= \left[r_0 + L_x L_t(t) \right]_+$$

where L_x corresponds to the first linear filter integral (over space x) and $L_t(t)$ corresponds to the second linear filter integral (over previous times τ).

Let's first define a parameter for the background rate. Then we will compute the integrals L_x and $L_t(t)$:

```
%RUN MODEL
r0 = 0; %background rate (ms^-1)
```

For now, let's assume there is zero background rate to these neurons. Next, let's compute the spatial integral L_x. This is simply a sum of the product $dx*D_x(x)*s_x(x)$, i.e. we just need to multiply $D_x(x)$ by $s_x(x)$ element by element, then multiply by dx, and then sum over all the spatial points for which neither of these are zero. (Technically, we could just multiply over x corresponding to the width of the target bar, but to keep our code general, we'll multiply over all elements of x for which the kernel $D_x(x)$ is non-negligible).

I.e., after all that explaining, we could do this with the line:

```
Target_L_x = sum(dx*D_x_vect.*Target_x_vect) %spatial linear filter integral
```

However, the act of multiplying corresponding elements of vectors and then summing over them is identical to matrix multiplying one vector by the other vector transposed:

```
Target_L_x = dx*D_x_vect*Target_x_vect' %spatial linear filter integral
```

This "matrix" multiplication of two vectors (where the second one must be transposed) is called the "*dot product*" or "*inner product*" of the two vectors. Try both of the above lines and check to see that they give the same result. Then remove the inelegant first line in favor of the nicer second line, and add a semicolon to suppress the output.

Now let's consider the $L_t(t)$ integral. This integral again looks like a "dot product", except that we need to do a separate dot product for each time t. Thus we will use a for loop to do this dot product for each time point and then assemble these values into a vector.

Recall that this integral weighs values of the stimulus a time τ ago by the weight $D_t(\tau)$. But how will we compute the stimulus at time t=0 when we don't have any values for the stimulus at previous times? The answer is, "We can't compute it without knowing values of the stimulus at previous times!!!". This gives us two choices:

1) Define the stimulus at times t<0 and note that these values will affect our neuron's firing rate at times t>0 up to the duration (i.e. length) of the kernel $D_t(\tau)$

2) Don't start computing the rate until times t sufficiently large enough that stimuli at times t<0 have negligible effect. Since our temporal receptive field extends backwards in time for a duration length(D_t_vect), we really can't start computing the rate until this time.

Either of the above choices is ok. Let's do the first one so that we have a firing rate at all times. First, let's set up a vector to hold the temporal linear filter result:

```
%prepare to do temporal filter integral
Target_L_t_vect = zeros(1,length(t_vect)); %set up vector to hold temporal filter values
```

Next, let's define the stimulus to be zero for times t<0 and append this to the beginning of the present stimulus vector (currently the Target_t_vect) as a new vector we will call Target_t_vect_long:

```
Stim_t_NegTimes = zeros(1,tau_vect_max+1); %need to add stimulus values at t<0 so
                        %can get what influenced cell at times t<tau_vect_max
Target_t_vect_long = [Stim_t_NegTimes Target_t_vect]; %includes Stimulus at times t<0
```

Now we are ready to do our for loop of dot products of the temporal kernel and stimulus vector. Note that, in order to multiply the kernel $D_t(\tau)$ by corresponding values $s_t(t-\tau)$, we need to list the stimulus values in reverse order. This is done as follows:

```
i = 0;
for t=0:dt:tmax
   i = i+1;
   Target_L_t_vect(i)=dt*D_t_vect*Target_t_vect_long(i+tau_vect_max:-1:i)';
end
```

Notice that the Target_t_vect_long indices work backwards from recent times to times in the past so as to align with the corresponding values of D_t_vect (which keep track of time *into the past*

τ). Keeping track of all the starting indices and ending indices of the vectors requires painstakingly keeping track of the lengths of all vectors and which elements correspond to what points in time. I recommend pulling out a sheet of paper to keep track of the vectors, their lengths, and what points in real time their elements correspond to—it's extremely difficult to do this in your head!

Finally let's multiply $L_t(t)$ by L_x and add the background rate to get the firing rate before thresholding:

```
Target_r_vect_NoThresh = 1000*(r0 + Target_L_x*Target_L_t_vect);  %1000 converts to Hz
```

Next let's add the thresholding:

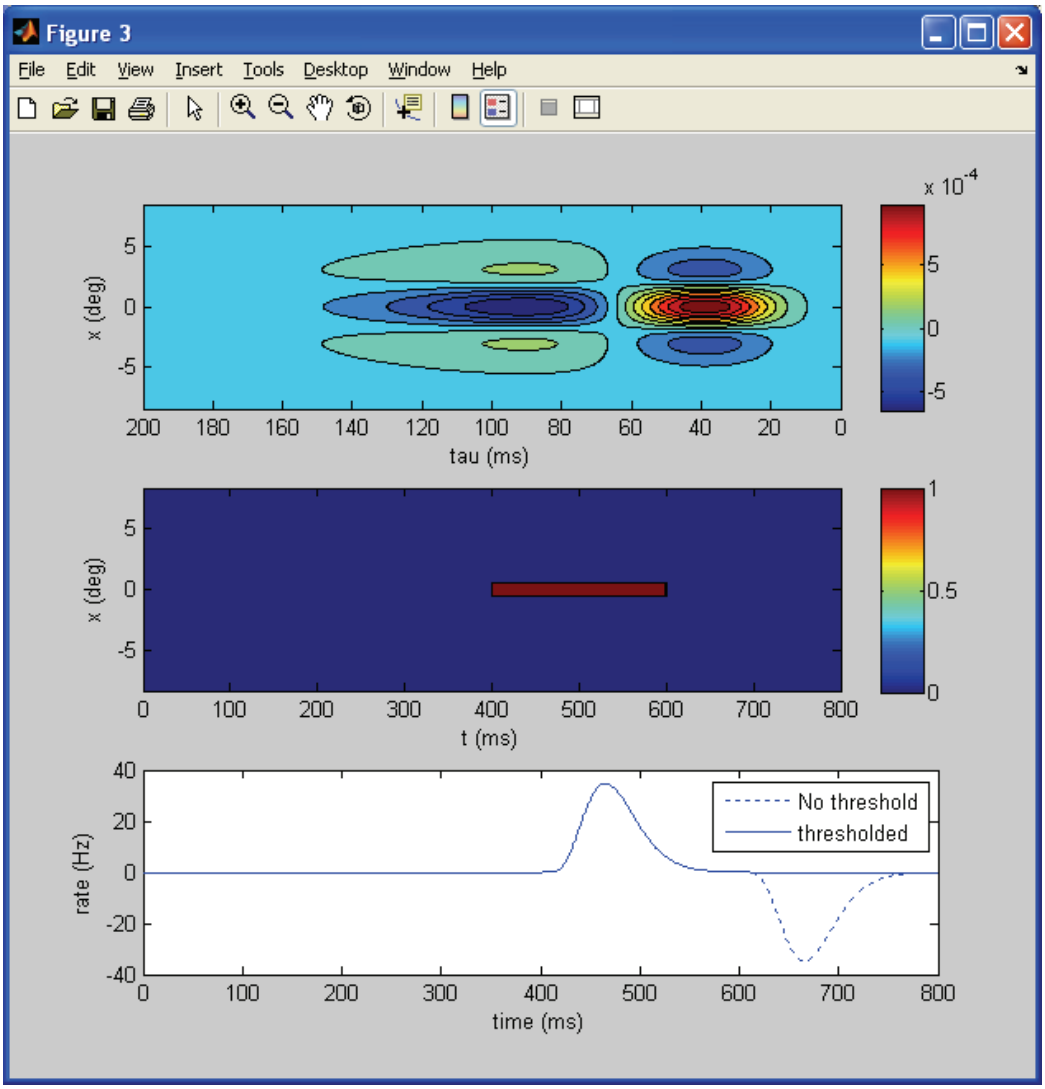
```
Target_r_vect = max(Target_r_vect_NoThresh,0); %thresholding
```

This gives thresholding as the maximum of the value of the first argument to the function and zero.

Finally, let's plot this and, to make things pretty, let's repeat our previous panels of the stimulus definition and the receptive field on the same plot:

```
%plot model results
figure(3)
subplot(3,1,1)
contourf(tau_vect,x_vect,D_xt_mat,12); %makes a contour plot of the data
colorbar
set(gca,'Xdir','reverse')
xlabel('tau (ms)')
ylabel('x (deg)')
subplot(3,1,2)
contourf(t_vect,x_vect,Target_xt_mat); %makes a contour plot of the data
colorbar
xlabel('t (ms)')
ylabel('x (deg)')
subplot(3,1,3)
plot(t_vect, Target_r_vect_NoThresh,':')
hold on;
plot(t_vect, Target_r_vect)
xlabel('time (ms)');
ylabel('rate (Hz)');
legend('No threshold', 'thresholded')
hold off;
```

You should get the following result:



The cell fires transiently when the stimulus first turns on, and then is actually driven negative (before thresholding) at the termination of the stimulus. Try running the model again with a background rate of r0 = 10e-3 ms^{-1} to see how the inhibitory offset pushes the cell below its background rate.

You should be able to figure out that this rate is qualitatively correct by mentally sliding the receptive field rightward across the Target stimulus and noting at what times it overlaps positive stimulus on positive receptive field versus positive stimulus on negative receptive field.

To summarize, in this section you should have added the following code to run the model with the target stimulus:

```
%RUN MODEL
r0 = 10e-3; %background rate (ms^-1)
Target_L_x = dx*D_x_vect*Target_x_vect'; %spatial linear filter integral

%prepare to do temporal filter integral
```

```
Target_L_t_vect = zeros(1,length(t_vect)); %set up vector to hold temporal filter values
Stim_t_NegTimes = zeros(1,tau_vect_max+1); %need to add stimulus values at t<0 so
                                %can get what influenced cell at times t<tau_vect_max
Target_t_vect_long = [Stim_t_NegTimes Target_t_vect]; %includes Stimulus at times t<0
i = 0;
for t=0:dt:tmax
   i = i+1;
   Target_L_t_vect(i)=dt*D_t_vect*Target_t_vect_long(i+tau_vect_max:-1:i)';
end
Target_r_vect_NoThresh = 1000*(r0 + Target_L_x*Target_L_t_vect);  %1000 converts to Hz
Target_r_vect = max(Target_r_vect_NoThresh,0); %thresholding
%plot model results
figure(3)
subplot(3,1,1)
contourf(tau_vect,x_vect,D_xt_mat,12); %makes a contour plot of the data
colorbar
set(gca,'Xdir','reverse')
xlabel('tau (ms)')
ylabel('x (deg)')
subplot(3,1,2)
contourf(t_vect,x_vect,Target_xt_mat); %makes a contour plot of the data
colorbar
xlabel('t (ms)')
ylabel('x (deg)')
subplot(3,1,3)
plot(t_vect, Target_r_vect_NoThresh,':')
hold on;
plot(t_vect, Target_r_vect)
xlabel('time (ms)');
ylabel('rate (Hz)');
legend('No threshold', 'thresholded')
hold off;
```

IV. Add the mask stimulus

A) Response to the mask alone

We now will add two masking bars presented for 200 ms immediately before the target stimulus. To do this, let's first modify the stimulus parameters to add a masking stimulus. Under "spatial parameters", add after the Target definition:

```
Mask1_LeftEnd = -1.0;
Mask1_RightEnd = -0.5;
Mask2_LeftEnd = 0.5;
Mask2_RightEnd = 1.0;
Mask_x_vect = [zeros(1,(Mask1_LeftEnd - x_min)/dx) ... %nothing flashed here
               ones(1,((Mask1_RightEnd - Mask1_LeftEnd)/dx)+1) ... %Target position
               zeros(1,((Mask2_LeftEnd - Mask1_RightEnd)/dx)-1) ... %nothing flashed here
               ones(1,((Mask2_RightEnd - Mask2_LeftEnd)/dx)+1) ... %Target position
               zeros(1,(x_max - Mask2_RightEnd)/dx)]; %nothing flashed here
```

This defines the mask as two bars of the same width as the Target, and flanking the sides of the target.

Next, under “temporal parameters”, add the Mask’s timing as occurring right before the targets:

```
Mask_on = 200; %ms
Mask_off = 400; %ms
Mask_t_vect = [zeros(1,Mask_on/dt) ...   %bar initially off from t=0 to t=Mask_on-dt
               ones(1,(Mask_off-Mask_on)/dt) ... %then Mask on
               zeros(1,((tmax-Mask_off)/dt)+1)];  %then Mask off again
```

Finally for this section of the code, under “define and plot stimulus in both space & time”, add:

```
Mask_xt_mat = Mask_x_vect'*Mask_t_vect;
```

Now that we have defined the mask, we are ready to run the model with the mask as the stimulus rather than with the target as the stimulus. First add under the corresponding locations to the analogous target lines:

```
Mask_L_x = dx*D_x_vect*Mask_x_vect';  %spatial linear filter integral

Mask_L_t_vect = zeros(1,length(t_vect)); %set up vector to hold temporal filter values

Mask_t_vect_long = [Stim_t_NegTimes Mask_t_vect]; %includes Stimulus at times t<0
```

Then in the for loop add below the Target line:

```
Mask_L_t_vect(i)=dt*D_t_vect*Mask_t_vect_long(i+tau_vect_max:-1:i)';
```

Finally let’s define rate vectors:

```
Mask_r_vect_NoThresh = 1000*(r0 + Mask_L_x*Mask_L_t_vect);  %1000 converts to Hz
Mask_r_vect = max(Mask_r_vect_NoThresh,0); %thresholding
Mask_r_vect_NoThresh = 1000*(r0 + Mask_L_x*Mask_L_t_vect);  %1000 converts to Hz
Mask_r_vect = max(Mask_r_vect_NoThresh,0); %thresholding
```

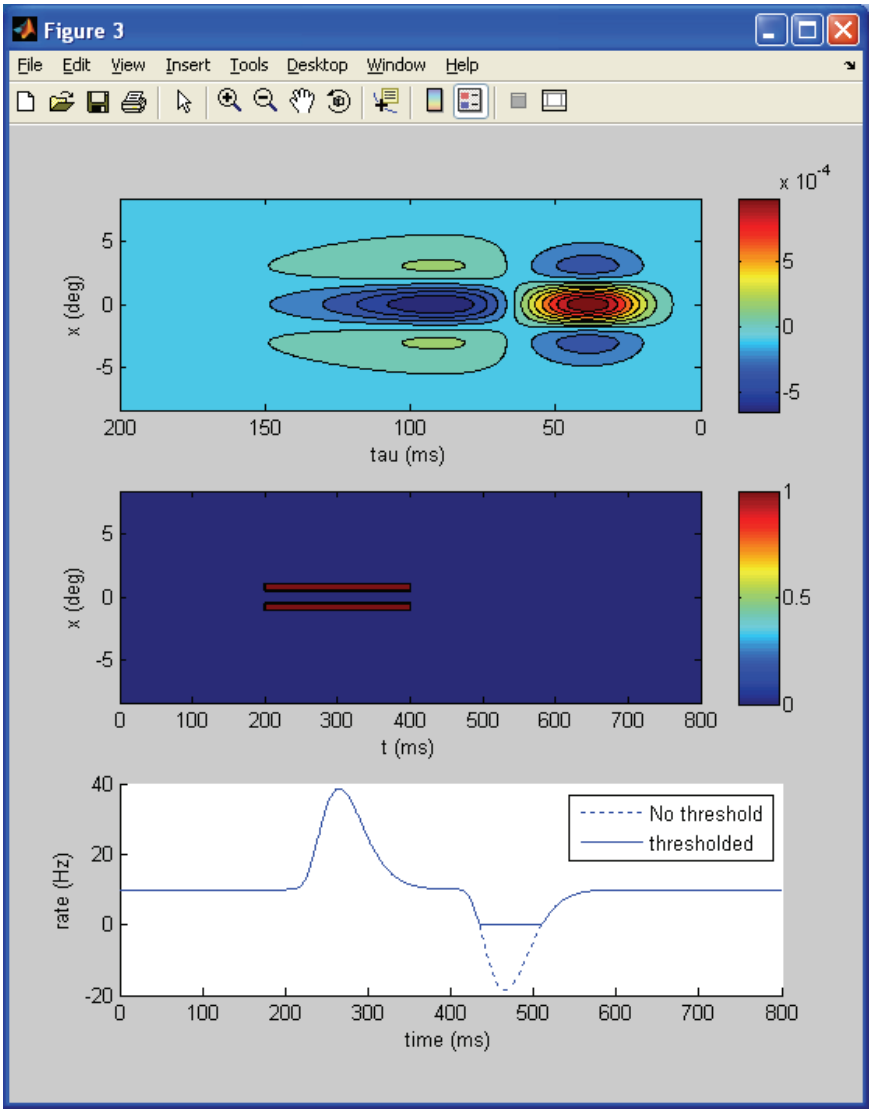
For the plots, in subplot(3,1,2) let’s comment out the lines for plotting the target and add lines for plotting the masks:

```
%contourf(t_vect,x_vect,Target_xt_mat); %makes a contour plot of the data
contourf(t_vect,x_vect,Mask_xt_mat); %makes a contour plot of the data

%plot(t_vect, Target_r_vect_NoThresh,':')
plot(t_vect,Mask_r_vect_NoThresh,':')

%plot(t_vect, Target_r_vect)
plot(t_vect,Mask_r_vect)
```

Run this and you should get a plot for the response to the mask alone, as follows:



You should verify that this makes sense for this stimulus and receptive field.

B) Response to both Mask and Target

Finally, we would like to look at the response to presentation of both the mask and the target in the same trial. To get this response, we will take advantage of the fact that linear filters are just summing over a product of the stimulus "pixels" $s(x,t-\tau)$ and the receptive field "pixels" $D(x,\tau)$. Thus, to get the linear filter portion of the response to the mask and the stimulus combined, i.e. to get $L_xL_t(t)$, we merely need to add the responses that we got to the mask alone and the target alone. We will then add the background rate and thresholding and obtain the final response.

Before doing this, let's make a space-time representation of the combined mask and stimulus by simply adding the two stimuli in the section "STIMULUS PARAMETERS: define and plot stimulus in both space & time". After the Mask_xt_mat line, add a line:

```
Both_xt_mat = Target_xt_mat + Mask_xt_mat; %mask and target both presented
```

Now let's go back to the "RUN MODEL" section and calculate the rate for both mask and target by using that the linear filter portions add, i.e. after the Mask_r_vect assignment line add:

```
Both_r_vect_NoThresh = 1000*(r0 + Target_L_x*Target_L_t_vect + Mask_L_x*Mask_L_t_vect);  %linear filter portions add linearly
Both_r_vect = max(Both_r_vect_NoThresh,0); %thresholding
```

Finally, let's adjust our plots to plot the result of both mask and target. First, let's revise the space-time stimulus by commenting out the previous plots in subplot(3,1,2) and adding this one:

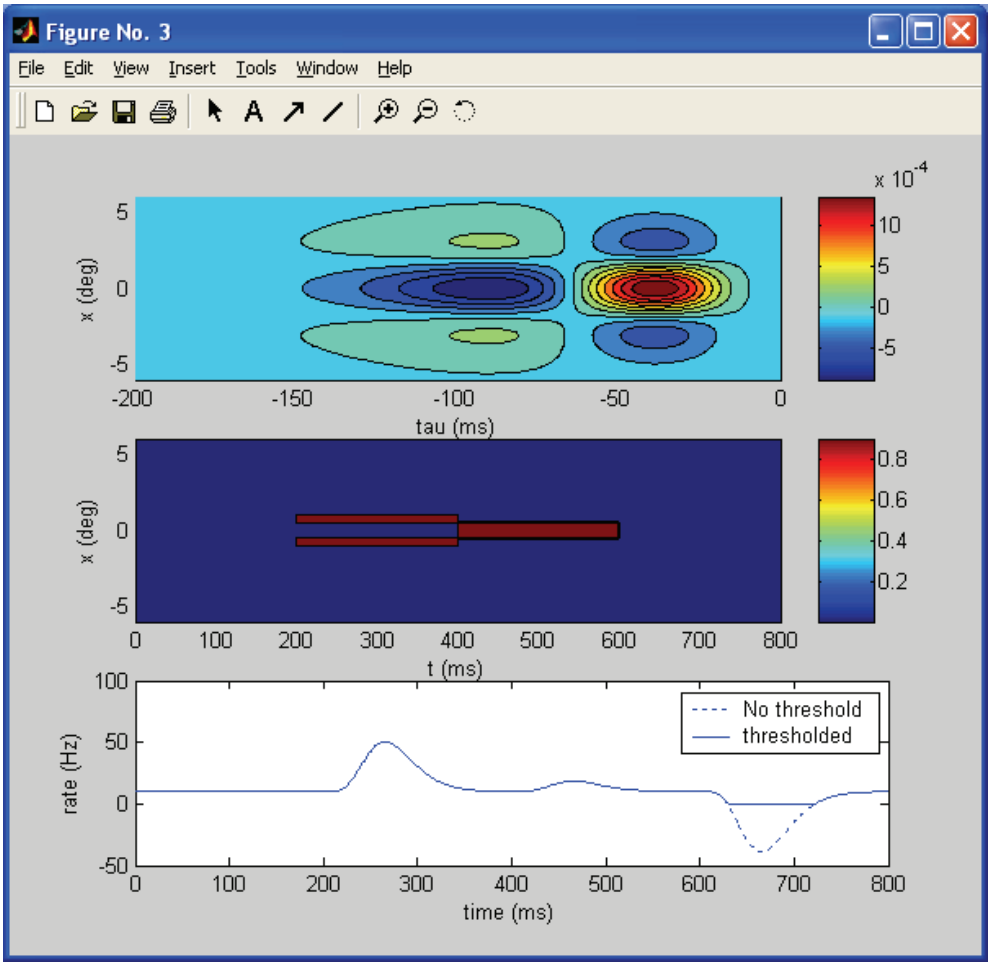
```
%contourf(t_vect,x_vect,Target_xt_mat); %makes a contour plot of the data
%contourf(t_vect,x_vect,Mask_xt_mat); %makes a contour plot of the data
contourf(t_vect,x_vect,Both_xt_mat); %makes a contour plot of the data
```

Next, let's similarly fix the two portions of subplot(3,1,3), as follows:

```
%plot(t_vect,Target_r_vect_NoThresh,':')
%plot(t_vect,Mask_r_vect_NoThresh,':')
plot(t_vect,Both_r_vect_NoThresh,':')

%plot(t_vect,Target_r_vect)
%plot(t_vect,Mask_r_vect)
plot(t_vect,Both_r_vect)
```

Run this and you should see that the response to the onset of the mask is still present but the response to the onset of the target has nearly disappeared—it has been masked!!!



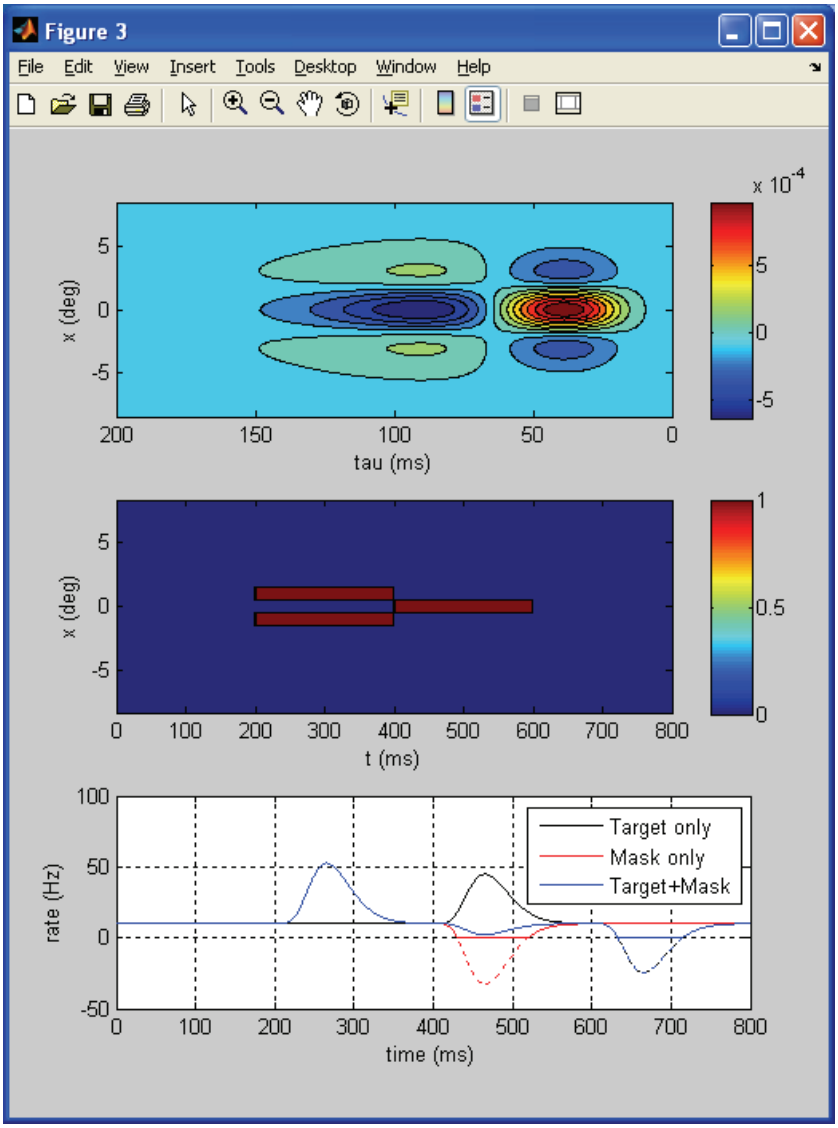
Why does this occur? (think about this before reading on...)

It occurs because the negative (inhibitory) offset-transient of the mask coincides with the positive onset-transient of the stimulus. We can see this more directly if we plot the responses to Target alone, Mask alone, and Both in the same plot but in different colors. Let's do black for the Target, red for the Mask, and blue for Both. We'll need to move up the 'hold on' statement, and let's also add a grid so that the final subplot(3,1,3) code reads:

```
subplot(3,1,3)
plot(t_vect,Target_r_vect_NoThresh,'k:')
hold on;
plot(t_vect,Mask_r_vect_NoThresh,'r:')
plot(t_vect,Both_r_vect_NoThresh,':')
p1 = plot(t_vect,Target_r_vect,'k');
p2 = plot(t_vect,Mask_r_vect,'r');
p3 = plot(t_vect,Both_r_vect);
xlabel('time (ms)');
ylabel('rate (Hz)');
legend([p1 p2 p3], 'Target only', 'Mask only', 'Target+Mask')
grid on
hold off;
```

The assignments of p1, p2, and p3 create variables called "**handles**" that can be used in other commands (in our case, the legend command) to tell MATLAB which plots we want to manipulate. Here, we create handles for the 3 plots that we want the legend to include and then pass these to the legend command as its first argument. Note that the order in which we give the plot handles corresponds to the order in which the 3 labels are assigned. If you plot this now, you can use your mouse to manually move the legend if it is located in a bad location.

Finally, in the illusion demo and experiments, each of the masks is actually equal in size to the target. We have been making the masks half the size of the targets. To fix this, modify the parameters Mask1_LeftEnd and Mask2_RightEnd to equal -1.5 and 1.5, respectively. Your final run should look like the following, with more masking of the target onset-response than when we had the thinner masks:



Before leaving this tutorial, you should play with other configurations of mask and target to see how they affect each other and whether you can predict their effects from analyzing the overlap of stimulus and response. For example, what would happen if the masks were the same size but wider apart? (Try it for various values. Explain what you see.)

Your final code (corresponding to the previous figure) should be:

```
%model an LGN neuron receiving a target bar of light preceded by two
%masking bars of light that, if appropriately positioned in space and
%appropriately timed, will make the target bar appear less visible!

clear all;
close all;

%FILTER PARAMETERS
%spatial filter parameters
sigma_c = 1.4;  %width of center portion of spatial r.f. [degrees]
sigma_s = 2.1;  %determines width of surround portion of spatial r.f. [deg]
A_c = 1;  %strength of center portion of spatial r.f.
A_s = 0.9; %strength of surround portion of spatial r.f.
dx = 0.05; %resolution of spatial grid
x_min = -4*sigma_s; %x-value roughly giving -x border of r.f.
x_max = 4*sigma_s;  %x-value roughly giving +x border of r.f.
x_vect = x_min:dx:x_max; %values of x over which to compute D_x

%temporal filter parameters
alpha = 1/10; %determines length of temporal filter [ms^-1]
dt = 1; %ms
tau_vect_max = 20/alpha; %tau value giving border of temporal r.f.
tau_vect = 0:dt:tau_vect_max; %value of tau over which to compute D_t

%define and then plot spatial kernel D_x
D_x_center = A_c*exp(-(x_vect.^2)/(2*(sigma_c^2)))/sqrt(2*pi*sigma_c^2);
D_x_surround = A_s*exp(-(x_vect.^2)/(2*(sigma_s^2)))/sqrt(2*pi*sigma_s^2);
D_x_vect = D_x_center - D_x_surround;
figure(1)
subplot(3,1,1)
%plot(x_vect, D_x_center, 'r--') %plot center with red dashed lines
hold on
%plot(x_vect, D_x_surround, 'k--') %plot surround with black dashed lines
plot(x_vect, D_x_vect)  %plot full receptive field with solid blue line
xlabel('x (deg)')
ylabel('D_x')

%define and then plot temporal kernel D_t
D_t_vect = alpha*exp(-alpha*tau_vect).*((alpha*tau_vect).^5/(5*4*3*2) - ...
            (alpha*tau_vect).^7/(7*6*5*4*3*2));
subplot(3,1,2)
plot(tau_vect,D_t_vect)
set(gca,'XDir','reverse')
xlabel('tau (ms)')
ylabel('D_t')

%define and then plot the full spatio-temporal kernel D(x,tau)
D_xt_mat = D_x_vect'*D_t_vect; %full 2-D r.f., with x as 1st dimension & t as 2nd dimension
subplot(3,1,3)
contour(tau_vect,x_vect,D_xt_mat,12); %makes a contour plot of the data
colorbar
```

```
set(gca,'Xdir','reverse')
xlabel('tau (ms)')
ylabel('x (deg)')

%STIMULUS PARAMETERS
%spatial parameters & plot of spatial locations of stimuli
Target_LeftEnd = -0.5; %position of start of bar [deg]
Target_RightEnd = 0.5; %position of end of bar [deg]
Target_x_vect = [zeros(1,(Target_LeftEnd - x_min)/dx)... %nothing flashed here
                 ones(1,((Target_RightEnd - Target_LeftEnd)/dx)+1)... %Target position
                 zeros(1,(x_max - Target_RightEnd)/dx)]; %nothing flashed here
Mask1_LeftEnd = -1.5;
Mask1_RightEnd = -0.5;
Mask2_LeftEnd = 0.5;
Mask2_RightEnd = 1.5;
Mask_x_vect = [zeros(1,(Mask1_LeftEnd - x_min)/dx) ... %nothing flashed here
               ones(1,((Mask1_RightEnd - Mask1_LeftEnd)/dx)+1) ... %Target position
               zeros(1,((Mask2_LeftEnd - Mask1_RightEnd)/dx)-1) ... %nothing flashed here
               ones(1,((Mask2_RightEnd - Mask2_LeftEnd)/dx)+1) ... %Target position
               zeros(1,(x_max - Mask2_RightEnd)/dx)]; %nothing flashed here
figure(2)
subplot(3,1,1)
plot(x_vect,Target_x_vect,'o')
xlabel('x (deg)')
ylabel('Target(1=Lt,0=Dk)')
%temporal parameters & plot of temporal locations of stimuli
tmax = 800; %ms
Target_on = 400; %ms
Target_off = 600; %ms
t_vect = 0:dt:tmax;
Target_t_vect = [zeros(1,Target_on/dt) ...   %bar initially off from t=0 to t=Target_on-dt
          ones(1,(Target_off-Target_on)/dt) ... %then Target on
          zeros(1,((tmax-Target_off)/dt)+1)];  %then Target off again
Mask_on = 200; %ms
Mask_off = 400; %ms
Mask_t_vect = [zeros(1,Mask_on/dt) ...   %bar initially off from t=0 to t=Mask_on-dt
               ones(1,(Mask_off-Mask_on)/dt) ... %then Mask on
               zeros(1,((tmax-Mask_off)/dt)+1)];  %then Mask off again
subplot(3,1,2)
plot(t_vect,Target_t_vect)
xlabel('t (ms)')
ylabel('Target')
%define and plot stimulus in both space & time
Target_xt_mat = Target_x_vect'*Target_t_vect;
Mask_xt_mat = Mask_x_vect'*Mask_t_vect;
Both_xt_mat = Target_xt_mat + Mask_xt_mat; %mask and target both presented
subplot(3,1,3)
contourf(t_vect,x_vect,Target_xt_mat); %makes a contour plot of the data
colorbar
xlabel('t (ms)')
ylabel('x (deg)')

%RUN MODEL
r0 = 10e-3; %background rate (ms^-1)
Target_L_x = dx*D_x_vect*Target_x_vect'; %spatial linear filter integral
```

```
Mask_L_x = dx*D_x_vect*Mask_x_vect';  %spatial linear filter integral

%prepare to do temporal filter integral
Target_L_t_vect = zeros(1,length(t_vect)); %set up vector to hold temporal filter values
Mask_L_t_vect = zeros(1,length(t_vect)); %set up vector to hold temporal filter values
Stim_t_NegTimes = zeros(1,tau_vect_max+1); %need to add stimulus values at t<0 so
                                %can get what influenced cell at times t<tau_vect_max
Target_t_vect_long = [Stim_t_NegTimes Target_t_vect]; %includes Stimulus at times t<0
Mask_t_vect_long = [Stim_t_NegTimes Mask_t_vect]; %includes Stimulus at times t<0
i = 0;
for t=0:dt:tmax
    i = i+1;
    Target_L_t_vect(i)=dt*D_t_vect*Target_t_vect_long(i+tau_vect_max:-1:i)';
    Mask_L_t_vect(i)=dt*D_t_vect*Mask_t_vect_long(i+tau_vect_max:-1:i)';
end
Target_r_vect_NoThresh = 1000*(r0 + Target_L_x*Target_L_t_vect);  %1000 converts to Hz
Target_r_vect = max(Target_r_vect_NoThresh,0); %thresholding
Mask_r_vect_NoThresh = 1000*(r0 + Mask_L_x*Mask_L_t_vect);  %1000 converts to Hz
Mask_r_vect = max(Mask_r_vect_NoThresh,0); %thresholding
Both_r_vect_NoThresh = 1000*(r0 + Target_L_x*Target_L_t_vect + Mask_L_x*Mask_L_t_vect);  %linear
filter portions add linearly
Both_r_vect = max(Both_r_vect_NoThresh,0); %thresholding
%plot model results
figure(3)
subplot(3,1,1)
contourf(tau_vect,x_vect,D_xt_mat,12); %makes a contour plot of the data
colorbar
set(gca,'Xdir','reverse')
xlabel('tau (ms)')
ylabel('x (deg)')
subplot(3,1,2)
%contourf(t_vect,x_vect,Target_xt_mat); %makes a contour plot of the data
%contourf(t_vect,x_vect,Mask_xt_mat); %makes a contour plot of the data
contourf(t_vect,x_vect,Both_xt_mat); %makes a contour plot of the data
colorbar
xlabel('t (ms)')
ylabel('x (deg)')
subplot(3,1,3)
plot(t_vect,Target_r_vect_NoThresh,'k:')
hold on;
plot(t_vect,Mask_r_vect_NoThresh,'r:')
plot(t_vect,Both_r_vect_NoThresh,':')
p1 = plot(t_vect,Target_r_vect,'k');
p2 = plot(t_vect,Mask_r_vect,'r');
p3 = plot(t_vect,Both_r_vect);
xlabel('time (ms)');
ylabel('rate (Hz)');
legend([p1 p2 p3], 'Target only', 'Mask only', 'Target+Mask')
grid on
hold off;
```